

Contents

[App Service Documentation](#)

[Overview](#)

[About Web Apps](#)

[About App Service on Linux](#)

[About App Service Environments](#)

[Compare hosting options](#)

[Quickstarts](#)

[Create .NET Core app](#)

[Create .NET Framework app](#)

[Create Node.js app](#)

[Create PHP app](#)

[Create Java app](#)

[Create static HTML site](#)

[Run Windows container](#)

[Tutorials](#)

[App with DB](#)

[.NET Core with SQL DB](#)

[.NET with SQL DB](#)

[PHP with MySQL](#)

[Node.js with MongoDB](#)

[Customize Windows \(container\)](#)

[Access SQL DB securely](#)

[Host RESTful API](#)

[Map custom domain](#)

[Secure domain with SSL](#)

[Add CDN](#)

[Authenticate users](#)

[Samples](#)

[Azure CLI](#)

[Azure PowerShell](#)

[Resource Manager templates](#)

[Concepts](#)

[App Service plans](#)

[OS functionality](#)

[Deployment best practices](#)

[Security](#)

[Recommendations](#)

[Authentication and authorization](#)

[OS and runtime patching](#)

[Built-in security controls](#)

[Security overview](#)

[Networking features](#)

[Overview](#)

[Application Gateway with service endpoints](#)

[Inbound and outbound IPs](#)

[Virtual Network Integration](#)

[Hybrid connections](#)

[Traffic Manager integration](#)

[Local cache](#)

[Diagnostics](#)

[How-To guides](#)

[Configure app](#)

[Configure common settings](#)

[Configure PHP](#)

[Configure Java](#)

[Connect to on-prem resources](#)

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

[Deploy to Azure](#)

[Deploy the app](#)

[Run from package](#)

[Deploy ZIP or WAR](#)

- [Deploy via FTP](#)
- [Deploy via cloud sync](#)
- [Deploy continuously](#)
- [Deploy from local Git](#)
- [Deploy with GitHub Actions](#)
- [Deploy with template](#)
- [Set deployment credentials](#)
- [Create staging environments](#)
- [Resource Manager template guidance](#)
- [Map custom domain](#)
 - [Buy domain](#)
 - [Map domains with Traffic Manager](#)
 - [Migrate an active domain](#)
- [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](#)
 - [Restrict access](#)
 - [Use a managed identity](#)
 - [Reference secrets from Key Vault](#)
 - [Use SSL cert in code](#)
 - [Configure TLS mutual authentication](#)
- [Scale app](#)
 - [Scale up server capacity](#)
 - [Configure PremiumV2 tier](#)
 - [Scale out to multiple instances](#)
 - [High density hosting](#)

- [Monitor app](#)
 - [Quotas & alerts](#)
 - [Enable logs](#)
- [Manage app](#)
 - [Manage the hosting plan](#)
 - [Back up an app](#)
 - [Restore a backup](#)
 - [Restore a snapshot](#)
 - [Clone an app](#)
 - [Restore deleted App Service](#)
- [Move app](#)
 - [Move between regions](#)
 - [Move subscriptions](#)
- [Run background tasks](#)
 - [Create WebJobs](#)
 - [Develop WebJobs using VS](#)
 - [Get started with WebJobs SDK](#)
 - [Use WebJobs SDK](#)
- [Reference](#)
 - [Azure CLI](#)
 - [Azure PowerShell](#)
 - [REST API](#)
 - [Resource Manager template](#)
- [Resources](#)
 - [App Service Blog](#)
 - [Build your skills with Microsoft Learn](#)
 - [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](#)

[Troubleshoot domain and certificate issues](#)

[FAQ](#)

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

[Deployment FAQ](#)

[Open-source technologies FAQ](#)

[Configuration and management FAQ](#)

[IP address change](#)

[Inbound IP address](#)

[Outbound IP address](#)

[SSL address](#)

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.

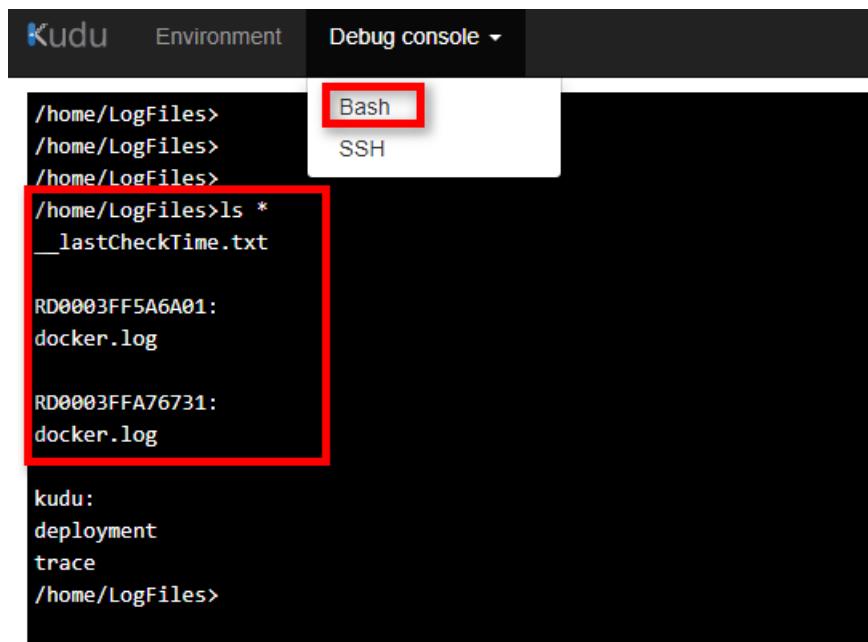
Troubleshooting

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](#).

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](#).

Create an ASP.NET Core web app in Azure

12/2/2019 • 4 minutes to read • [Edit Online](#)

NOTE

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

[Azure App Service](#) provides a highly scalable, self-patching web hosting service.

This quickstart shows how to deploy your first ASP.NET Core web app to Azure App Service. When you're finished, you'll have a resource group that consists of an App Service plan and an App Service app with a deployed web application.

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

Prerequisites

To complete this tutorial, install [Visual Studio 2019](#) with the **ASP.NET and web development** workload.

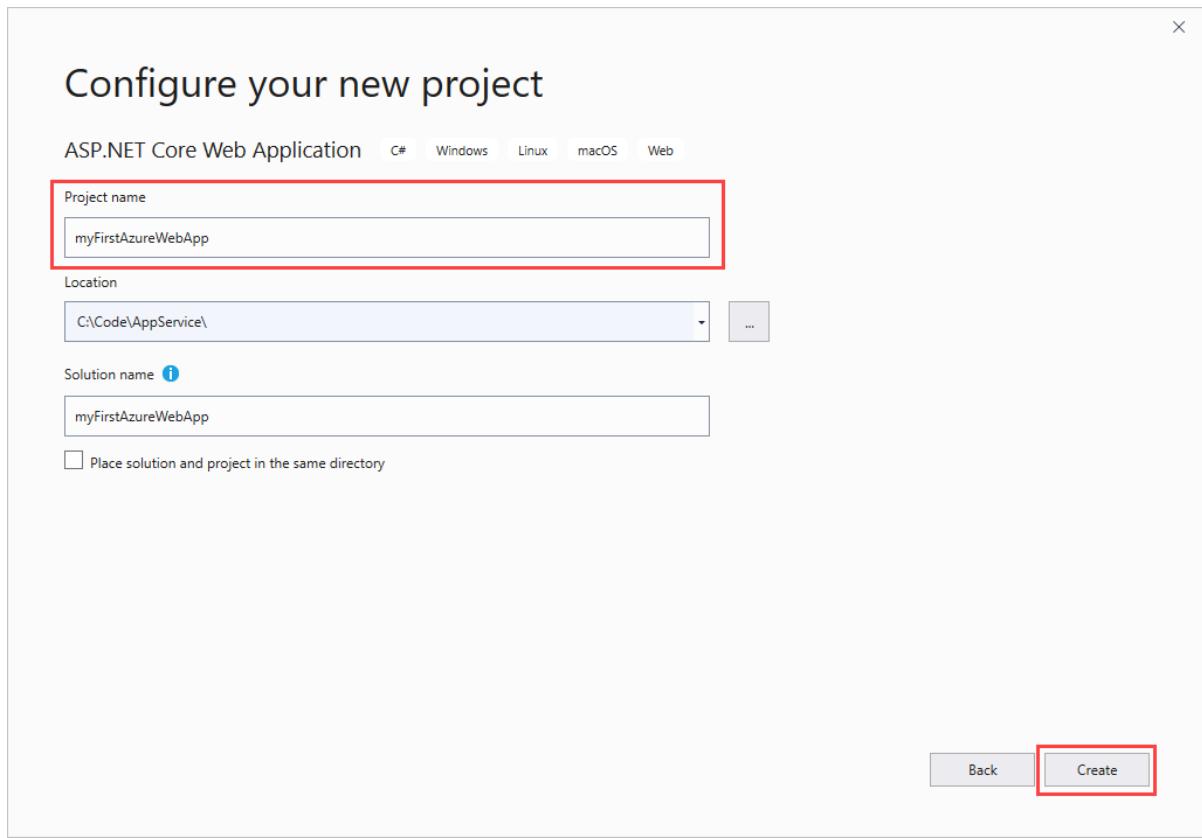
If you've installed Visual Studio 2019 already:

- Install the latest updates in Visual Studio by selecting **Help > Check for Updates**.
- Add the workload by selecting **Tools > Get Tools and Features**.

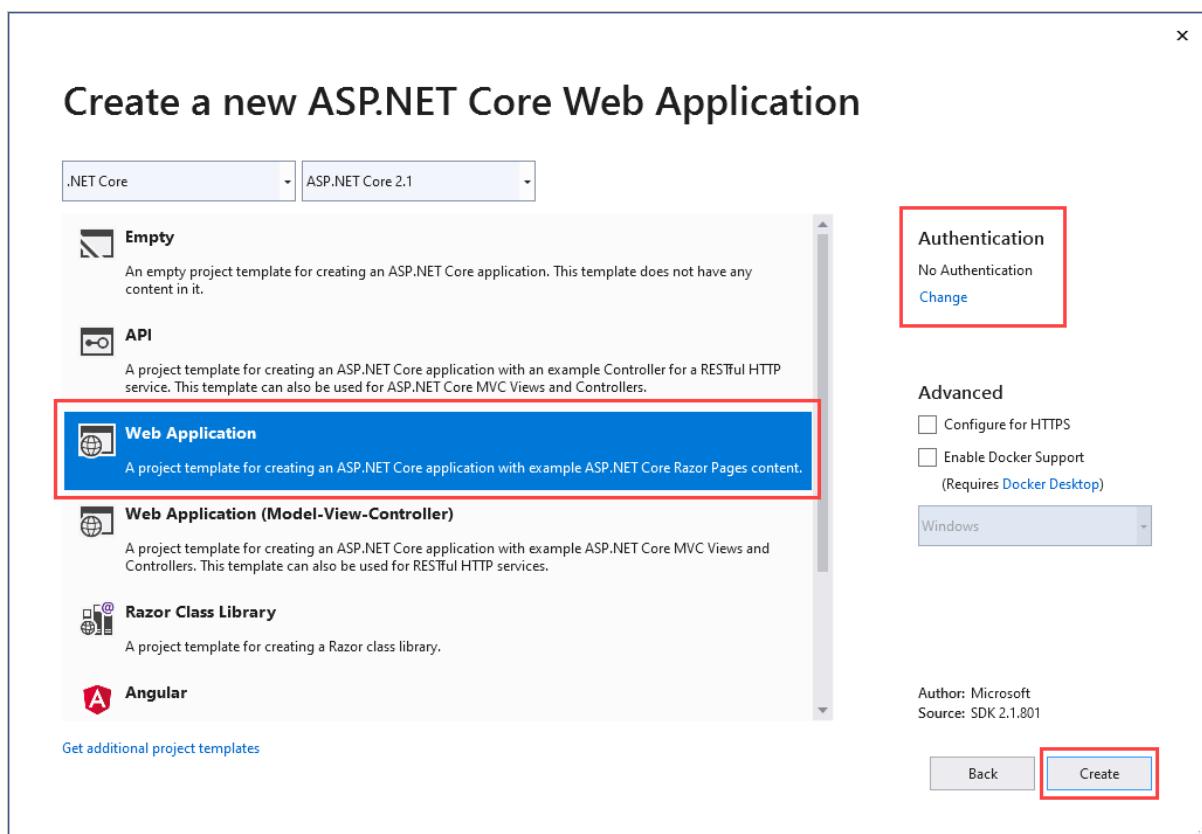
Create an ASP.NET Core web app

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

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

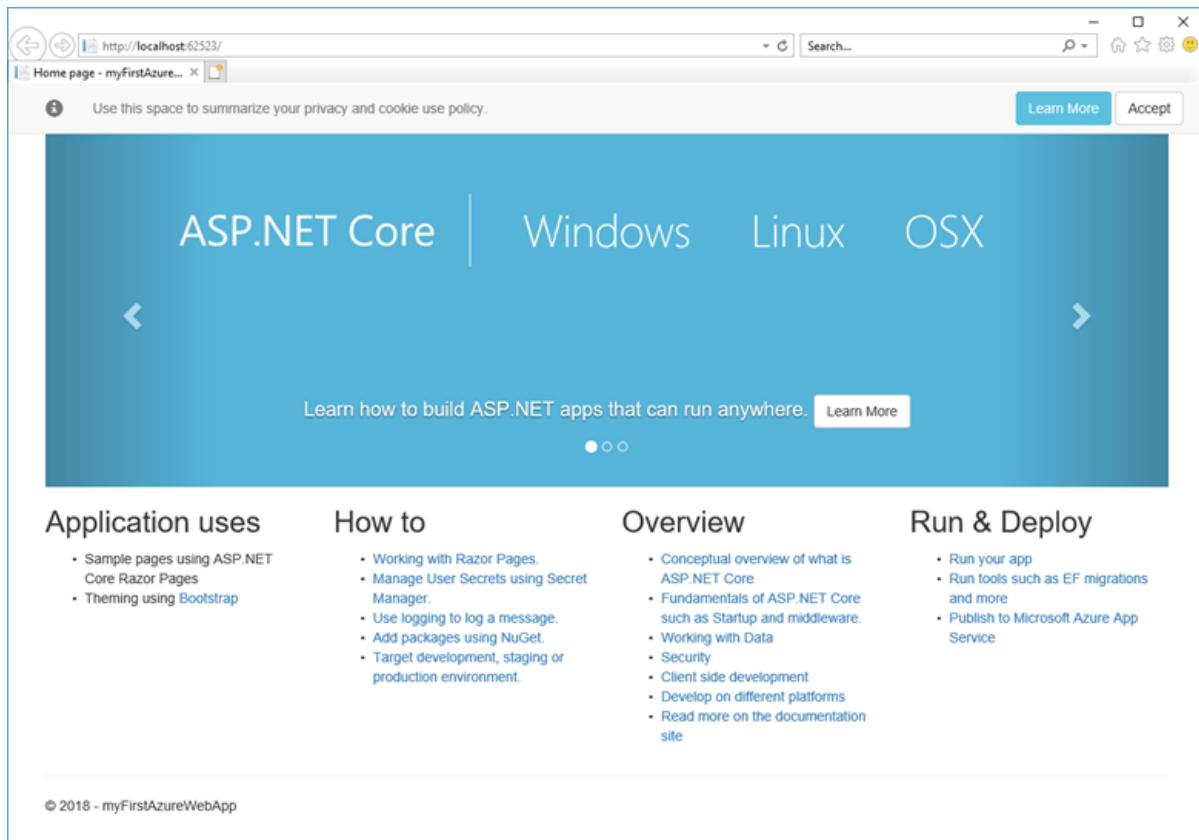


4. For this quickstart, choose the **Web Application** template. Make sure authentication is set to **No Authentication** and no other option is selected. Select **Create**.



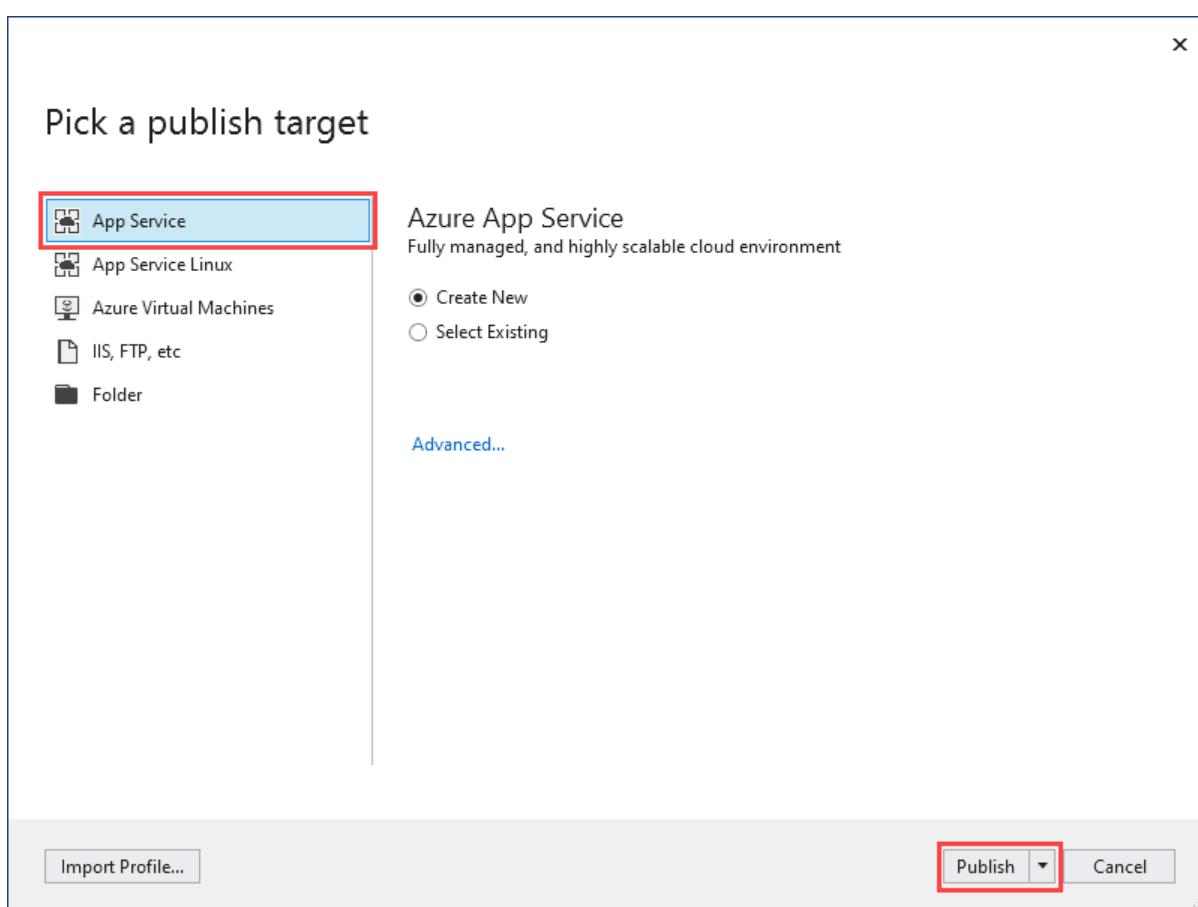
You can deploy any type of ASP.NET Core web app to Azure.

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



Publish your web app

1. In **Solution Explorer**, right-click the **myFirstAzureWebApp** project and select **Publish**.
2. Choose **App Service** and then select **Publish**.

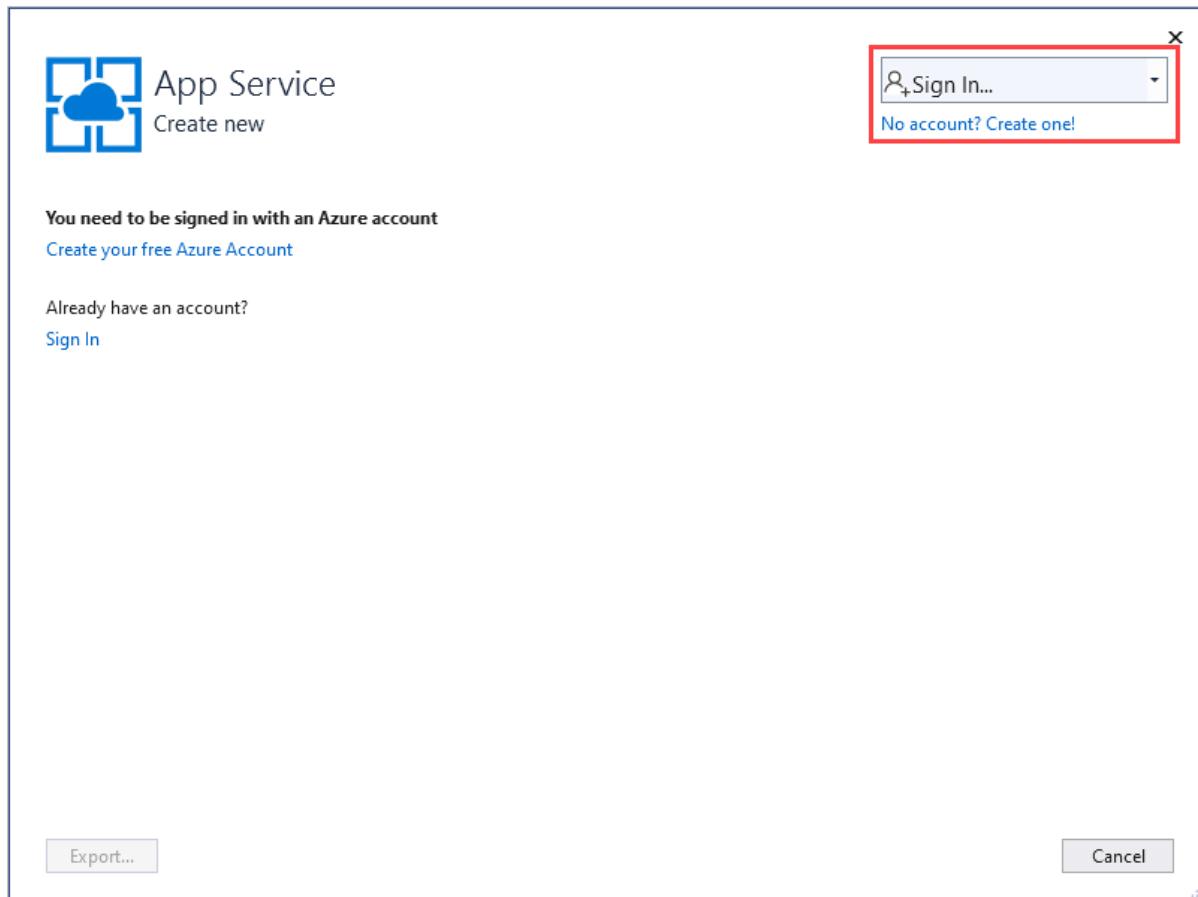


3. In **App Service Create new**, your options depend on whether you're signed in to Azure already and whether you have a Visual Studio account linked to an Azure account. Select either **Add an account** or

Sign in to sign in to your Azure subscription. If you're already signed in, select the account you want.

NOTE

If you're already signed in, don't select **Create** yet.



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.

4. For **Resource group**, select **New**.

5. In **New resource group name**, enter *myResourceGroup* and select **OK**.

An **App Service plan** specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

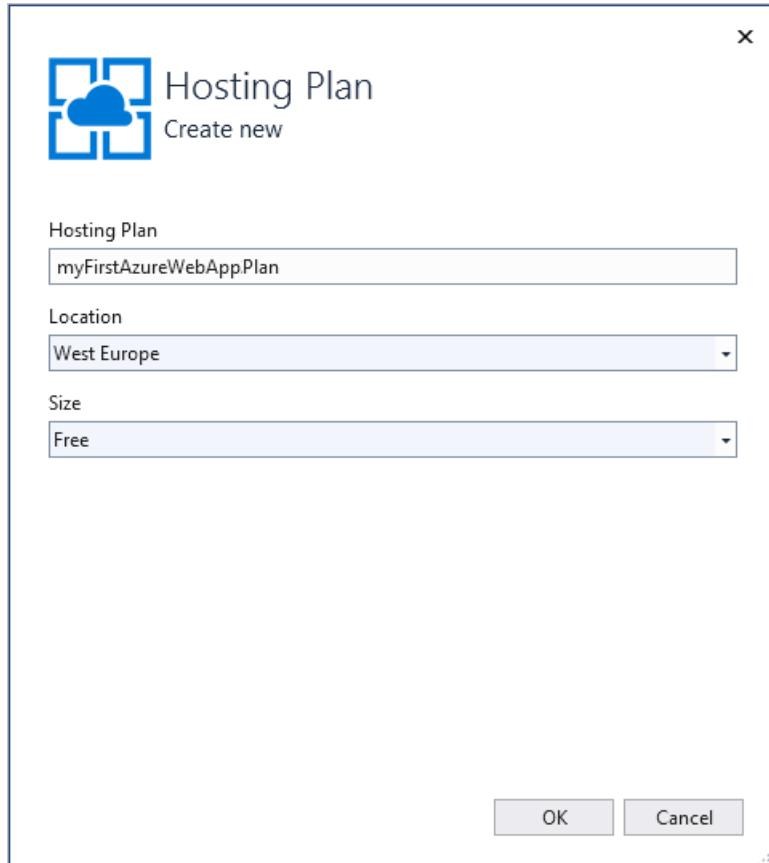
App Service plans define:

- Region (for example: North Europe, East US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 20 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

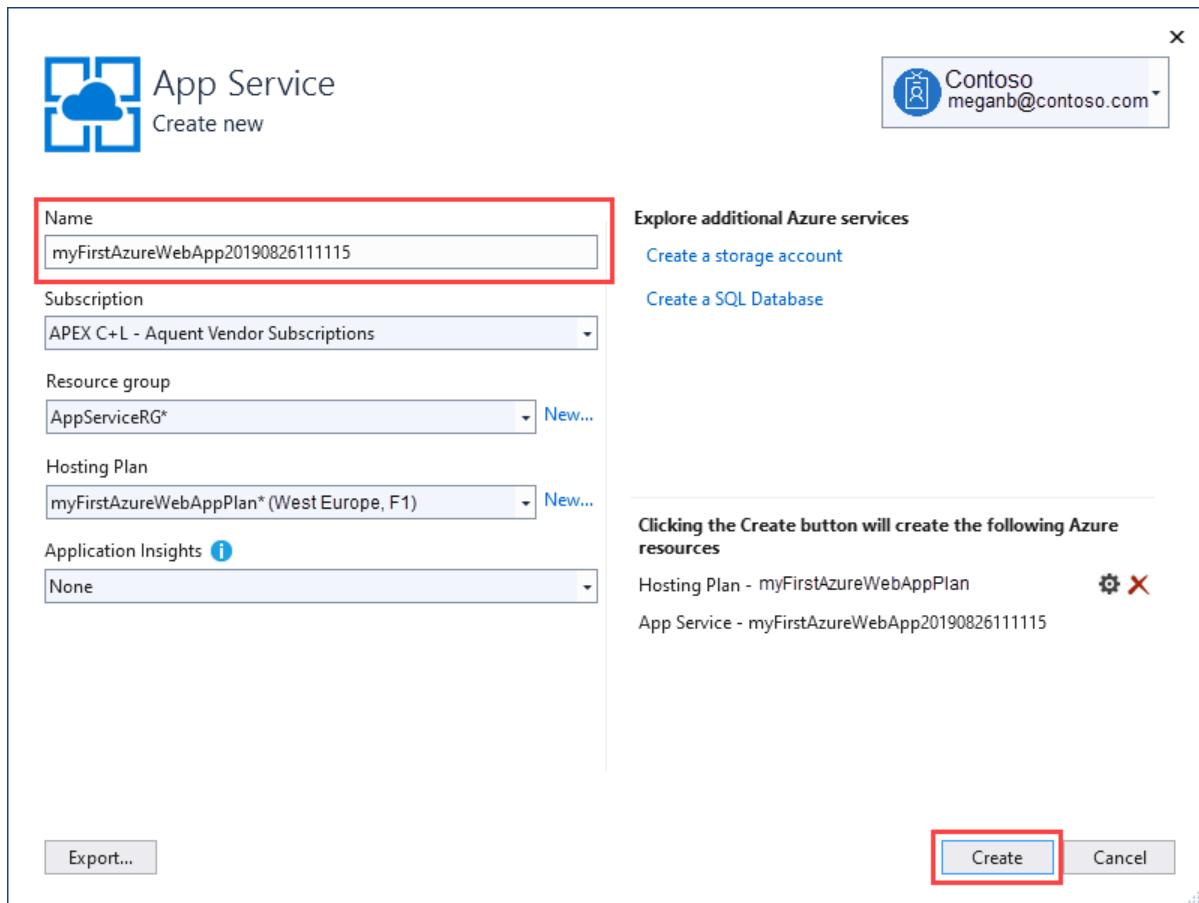
6. For the **Hosting Plan**, select **New**.

7. In the **Configure Hosting Plan** dialog, enter the values from the following table, and then select **OK**.

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

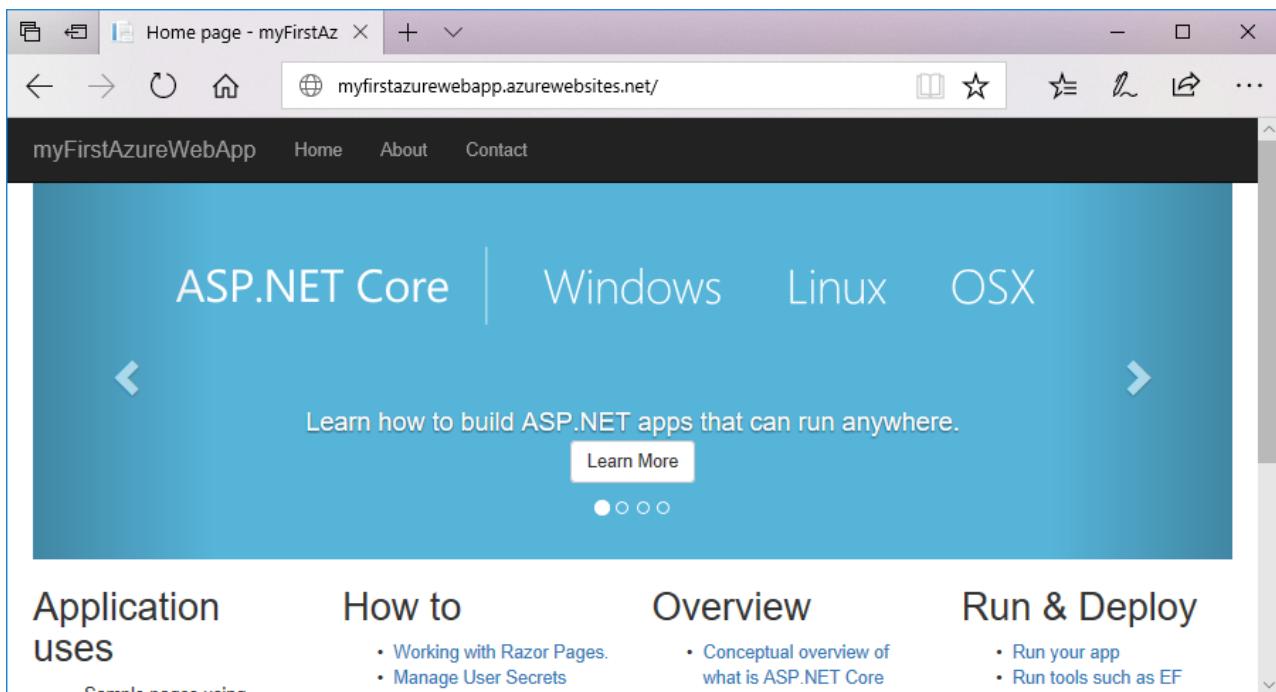


8. In **Name**, enter a unique app name that includes only the valid characters are `a-z`, `A-Z`, `0-9`, and `-`. You can accept the automatically generated unique name. The URL of the web app is `http://<app_name>.azurewebsites.net`, where `<app_name>` is your app name.



9. Select **Create** to start creating the Azure resources.

Once the wizard completes, it publishes the ASP.NET Core web app to Azure, and then launches the app in the default browser.



The app name specified in the **App Service Create new** page is used as the URL prefix in the format
`http://<app_name>.azurewebsites.net`.

Congratulations! Your ASP.NET Core web app is running live in Azure App Service.

Update the app and redeploy

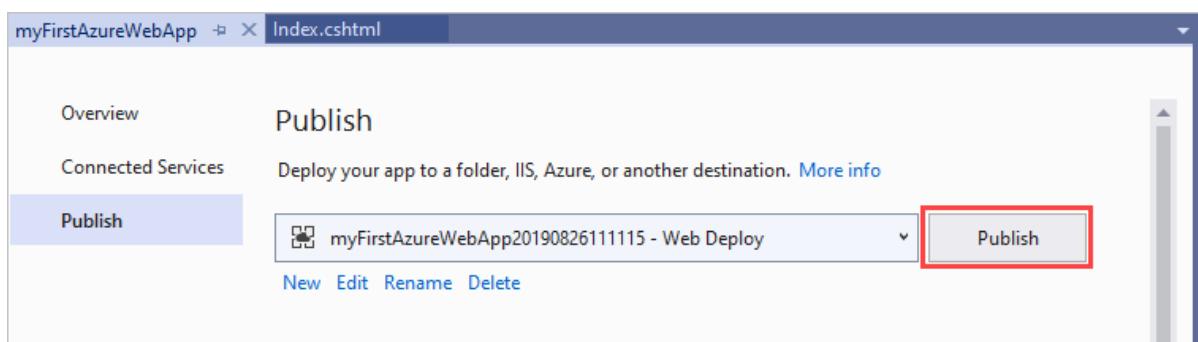
1. In **Solution Explorer**, under your project, open **Pages > Index.cshtml**.

2. Replace the two `<div>` tags with the following code:

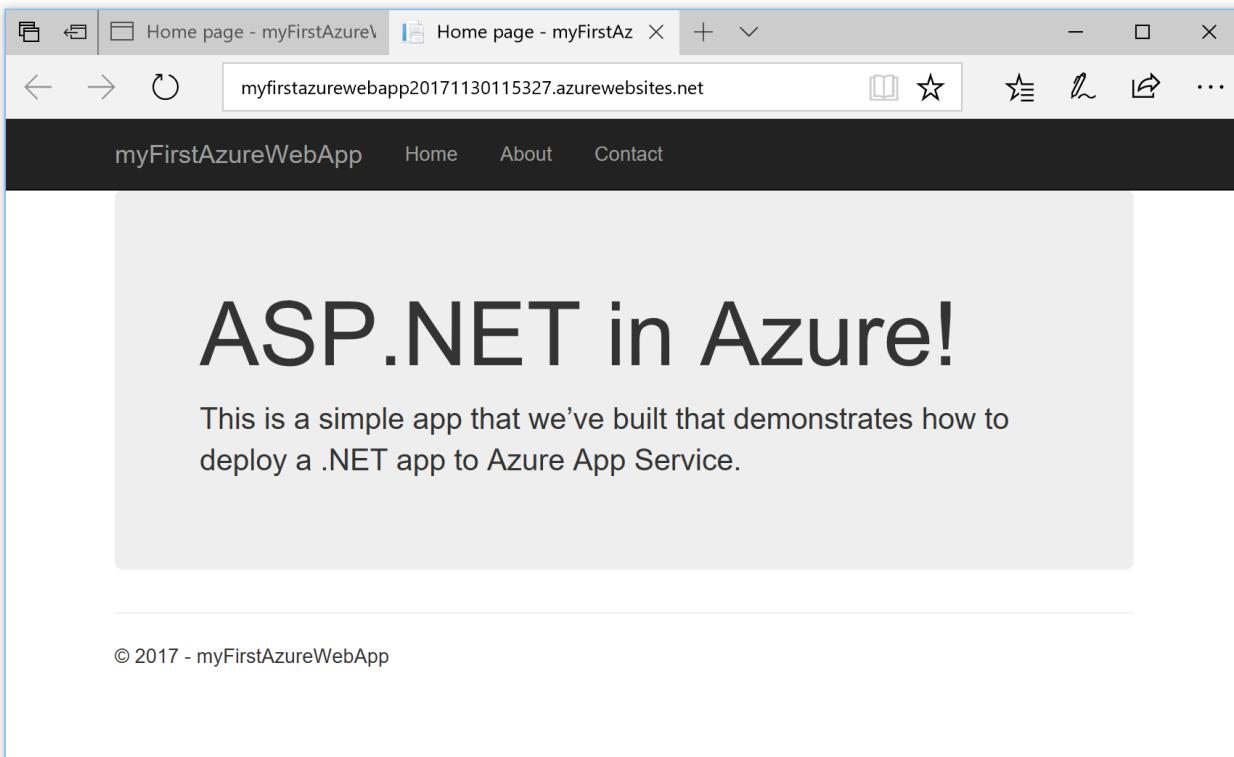
```
<div class="jumbotron">
    <h1>ASP.NET in Azure!</h1>
    <p class="lead">This is a simple app that we've built that demonstrates how to deploy a .NET app to
    Azure App Service.</p>
</div>
```

3. To redeploy to Azure, right-click the **myFirstAzureWebApp** project in **Solution Explorer** and select **Publish**.

4. In the **Publish** summary page, select **Publish**.



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



Manage the Azure app

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

The screenshot shows the Microsoft Azure portal interface. In the top left, there's a sidebar titled "Azure services" with options like "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". Below this is a section for "Recent resources" with a single item named "cephalin320170403020701". On the right, a search bar says "app services" and a list of services is shown, with "App Services" highlighted by a red box. Other listed services include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. Below the service list is a "Resources" section with a message "No results were found." At the bottom, there are navigation links for "Subscriptions", "Resource groups", and "All resources".

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

The screenshot shows the "App Services" blade in the Azure portal. The top navigation bar includes "Home > App Services" and "Documentation". Below the navigation is a toolbar with "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", and "More". A note says "Subscriptions: All 2 selected – Don't see a subscription? Open Directory + Subscription settings". There are filter buttons for "Filter by na...", "All subsc... ▾", "All resou... ▾", "All locati... ▾", "All tags ▾", and "No group... ▾". The main area displays a table with 6 items, each with a checkbox, name, status, app type, and app service plan. The row for "myFirstAzureWebApp20190..." is highlighted with a red box.

	Name	Status	App Type	App Service Plan
<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	ServiceP
<input type="checkbox"/>	WebApplicationASPDotNET...	Running	Web App	ServicePl

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

myFirstAzureWebApp20190826111115

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Deployment

Quickstart

Deployment slots

Deployment Center

Settings

Configuration

Authentication / Authorization

Resource group (change)
myResourceGroup

Status
Running

Location
West Europe

Subscription (change)
Contoso Subscription

Subscription ID
89d520cd-eb62-4f56-88a3-43087e91c288

Tags (change)
Click here to add tags

Click here to access Application Insights for monitoring and profiling for your ASP.NET Core app. →

Diagnose and solve problems

Application Insights

App Service Advisor

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, you can delete them by deleting the resource group.

From the Azure portal menu or **Home** page, select **Resource groups**, and on the **Resource groups** page, select **myResourceGroup**.

On the **myResourceGroup** page, make sure that the listed resources are the ones you want to delete.

Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

Next steps

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

Create an ASP.NET Framework web app in Azure

12/2/2019 • 3 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service.

This quickstart shows how to deploy your first ASP.NET web app to Azure App Service. When you're finished, you'll have an App Service plan. You'll also have an App Service app with a deployed web application.

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

Prerequisites

To complete this tutorial, install [Visual Studio 2019](#) with the **ASP.NET and web development** workload.

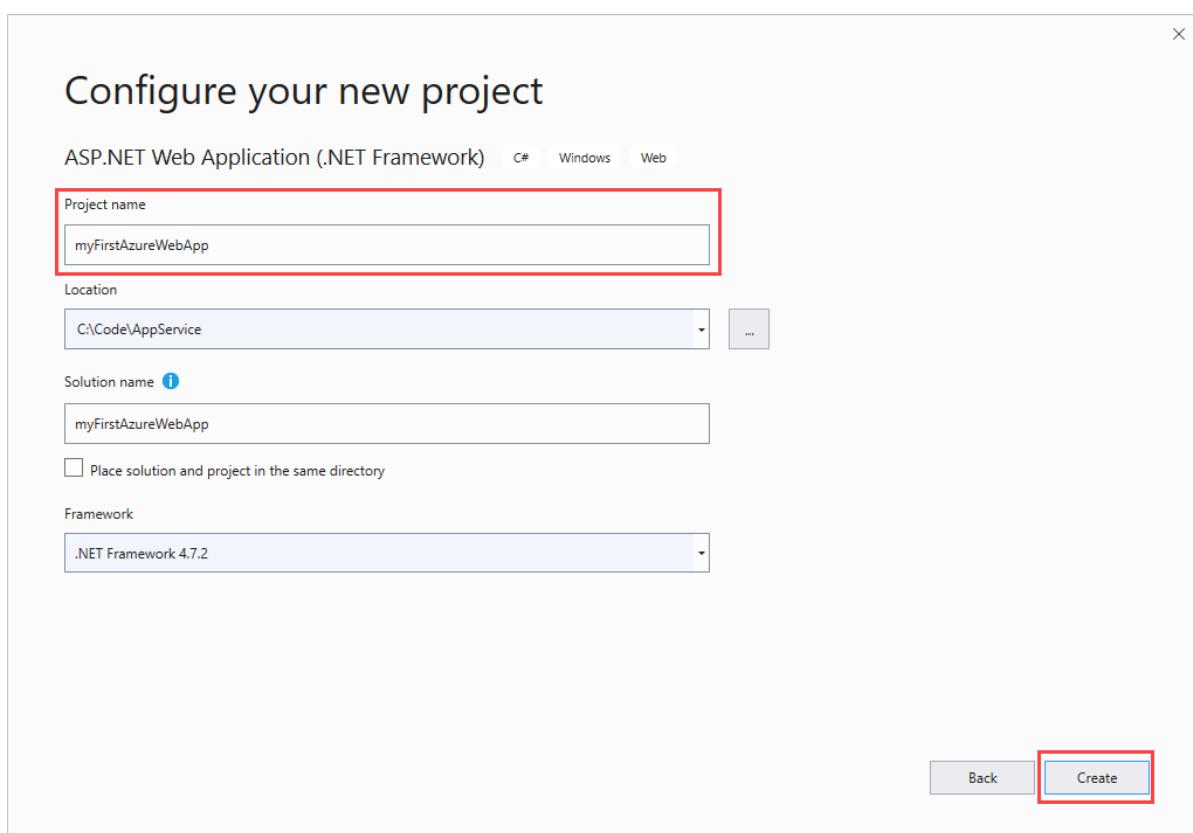
If you've installed Visual Studio 2019 already:

- Install the latest updates in Visual Studio by selecting **Help > Check for Updates**.
- Add the workload by selecting **Tools > Get Tools and Features**.

Create an ASP.NET web app

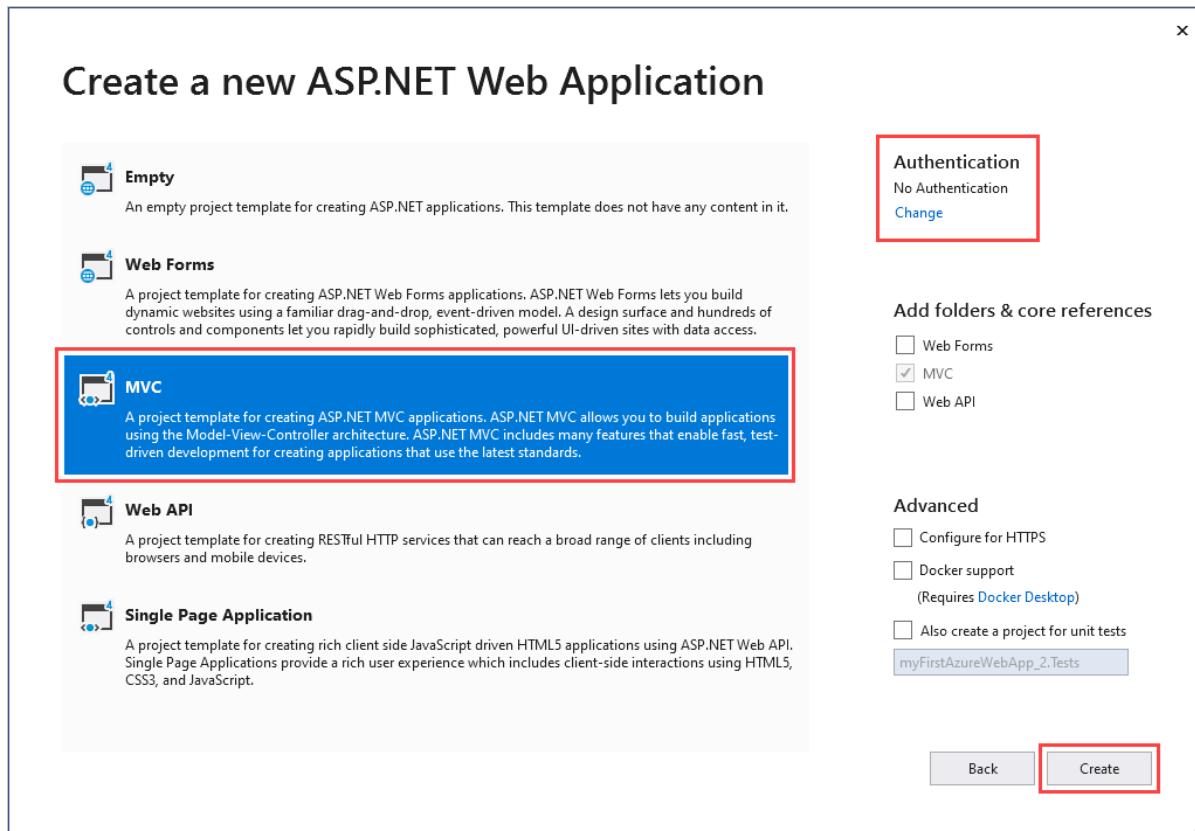
Create an ASP.NET web app by following these steps:

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

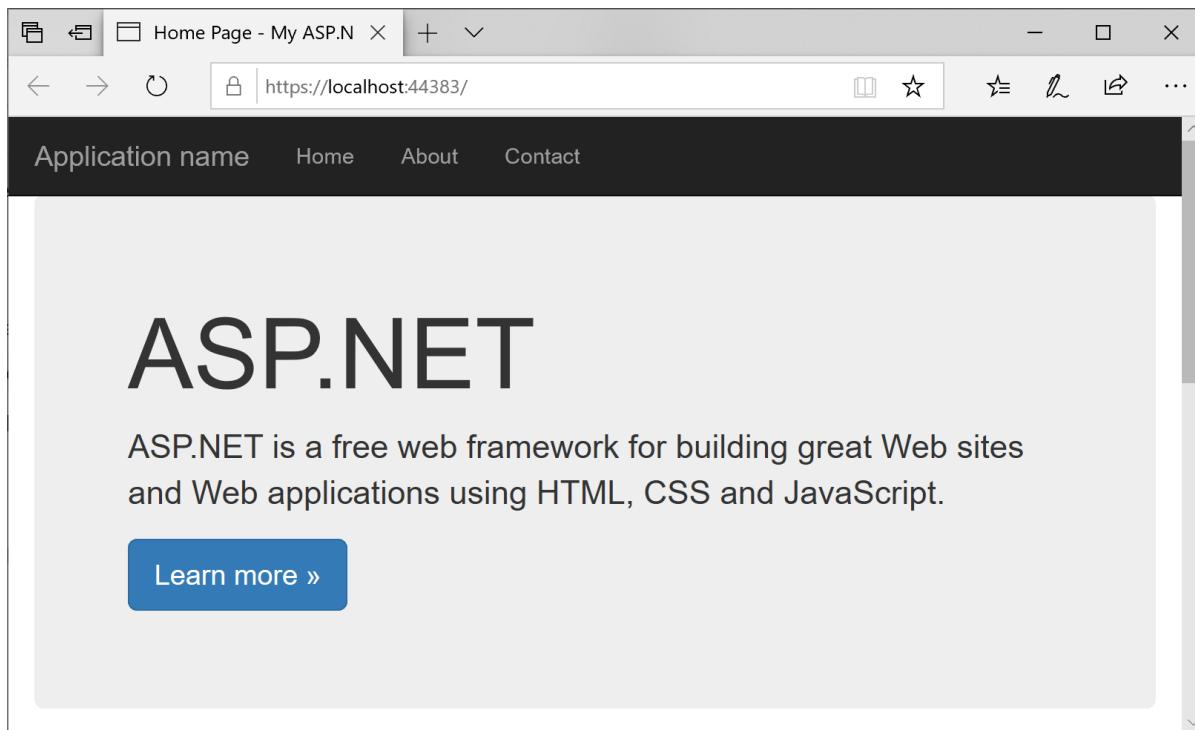


4. You can deploy any type of ASP.NET web app to Azure. For this quickstart, choose the **MVC** template.

5. Make sure authentication is set to **No Authentication**. Select **Create**.

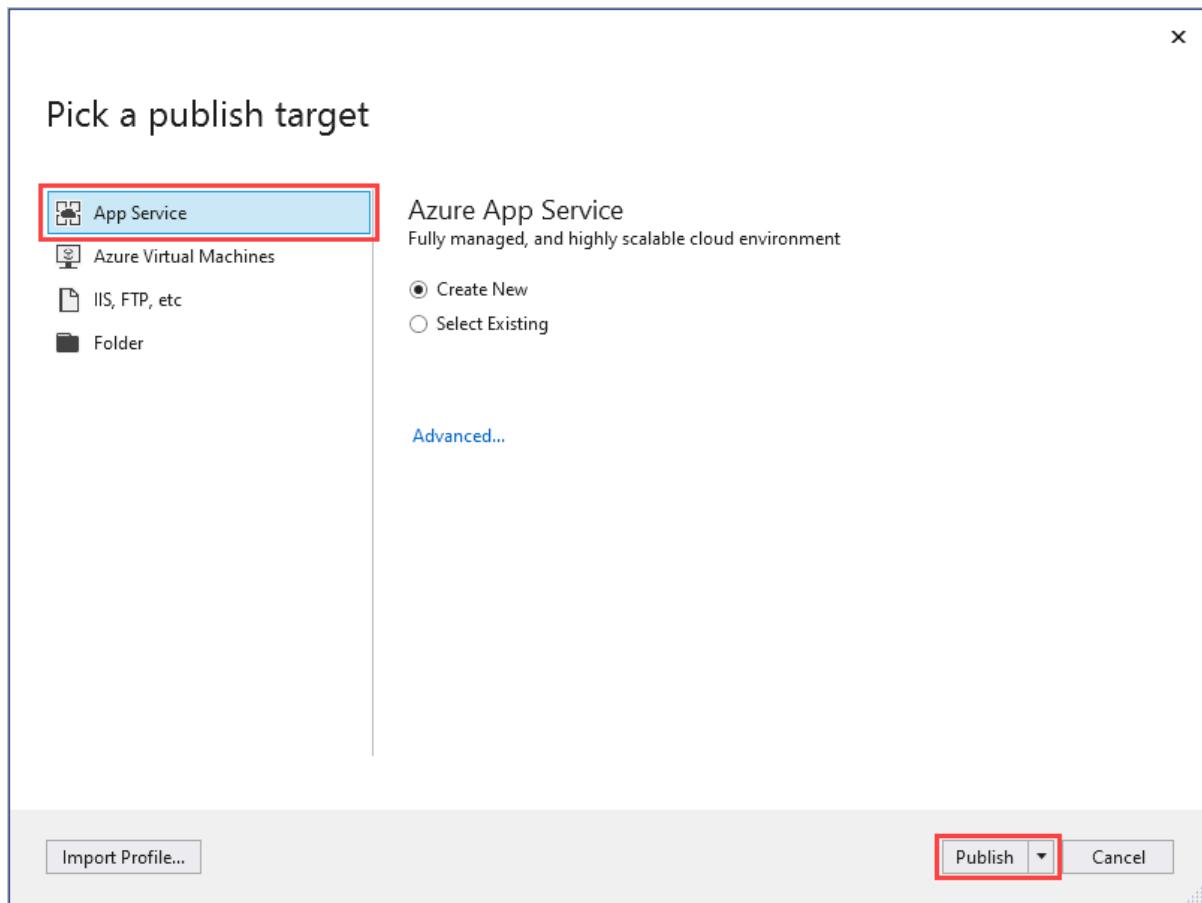


6. From the Visual Studio menu, select **Debug > Start Without Debugging** to run the web app locally.



Publish your web app

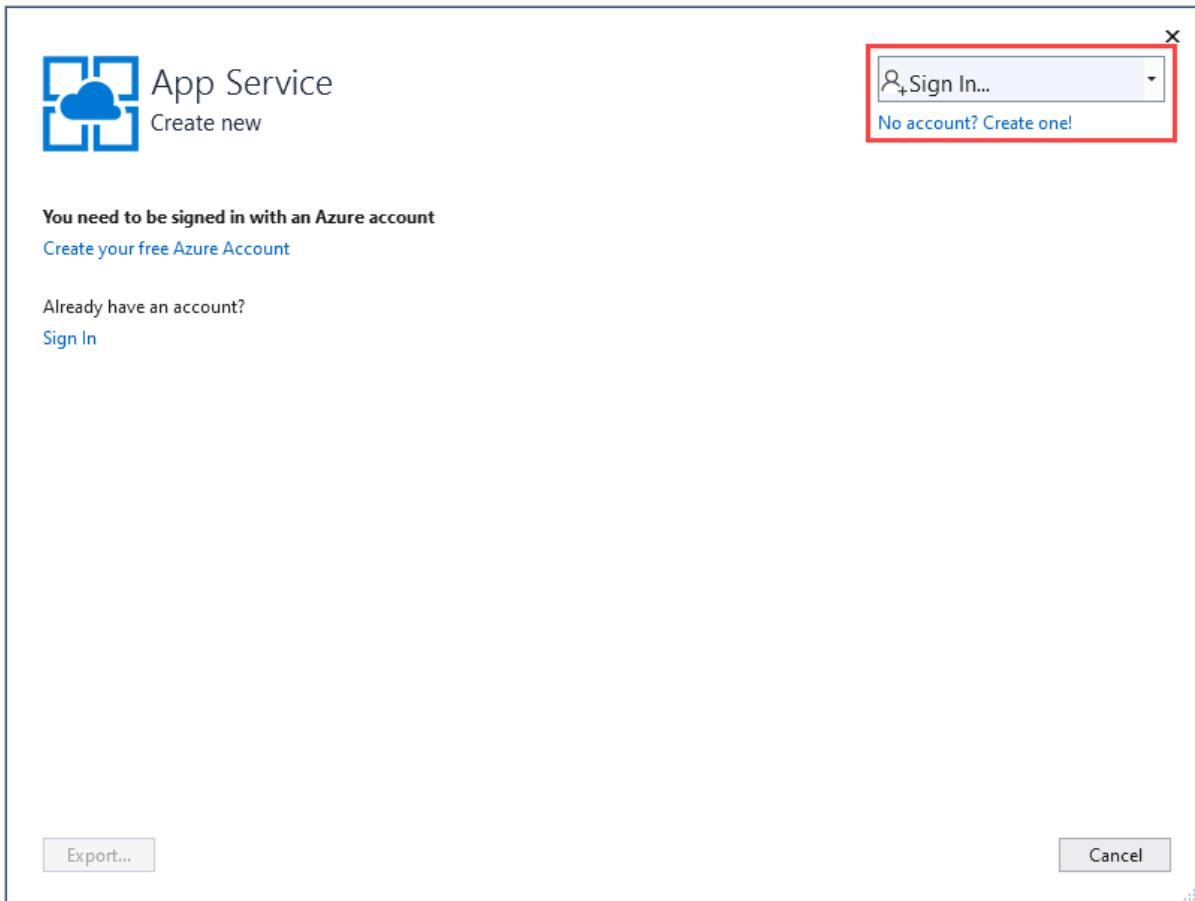
1. In **Solution Explorer**, right-click the **myFirstAzureWebApp** project and select **Publish**.
2. Choose **App Service** and then change **Create profile** to **Publish**.



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

NOTE

If you're already signed in, don't select **Create** yet.



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.

4. For **Resource group**, select **New**.

5. In **New resource group name**, enter *myResourceGroup* and select **OK**.

An [App Service plan](#) specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

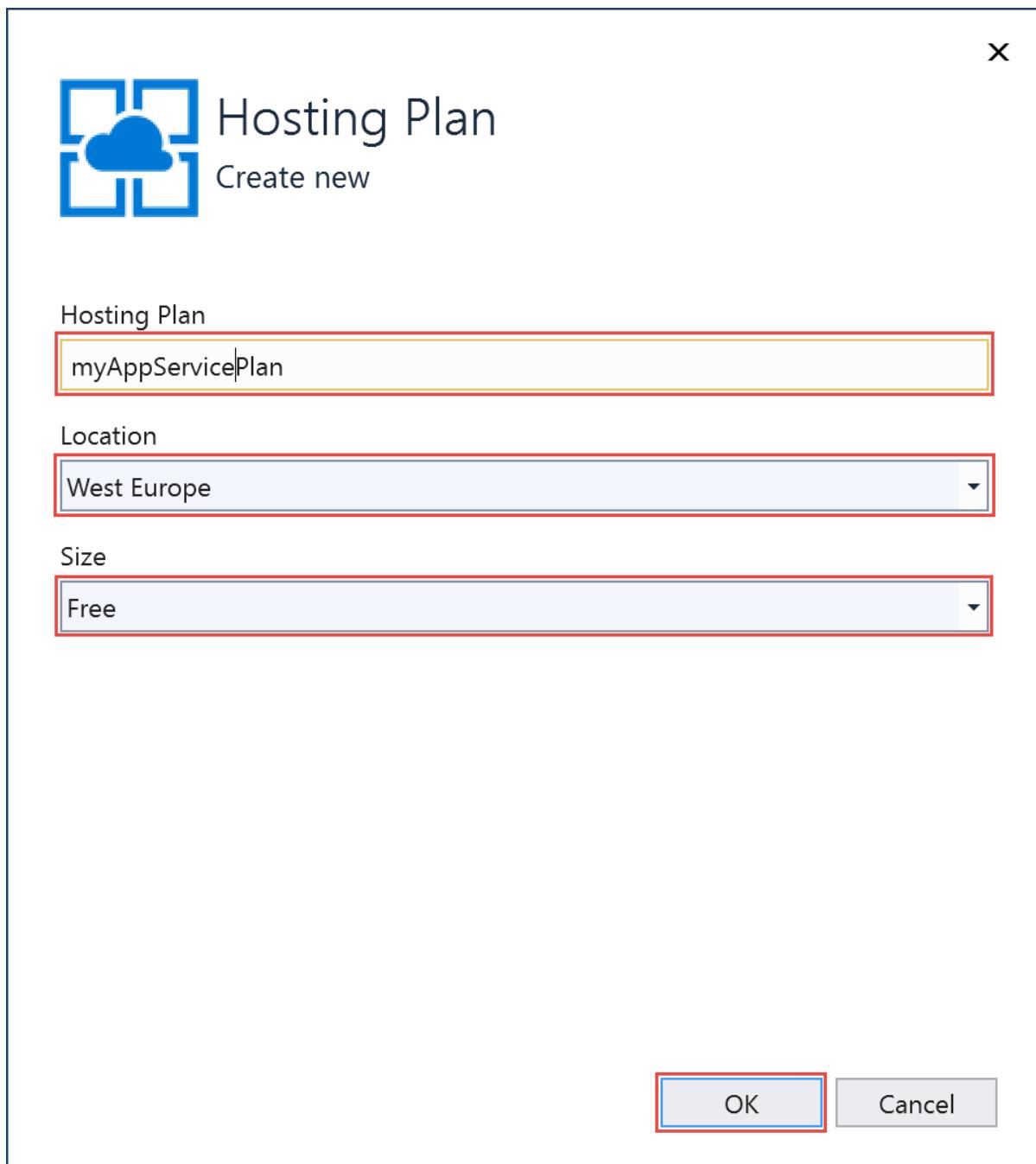
App Service plans define:

- Region (for example: North Europe, East US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 20 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

6. For **Hosting Plan**, select **New**.

7. In the **Configure Hosting Plan** dialog, enter the values from the following table, and then select **OK**.

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



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

 App Service
Create new

Name

Subscription

Resource group

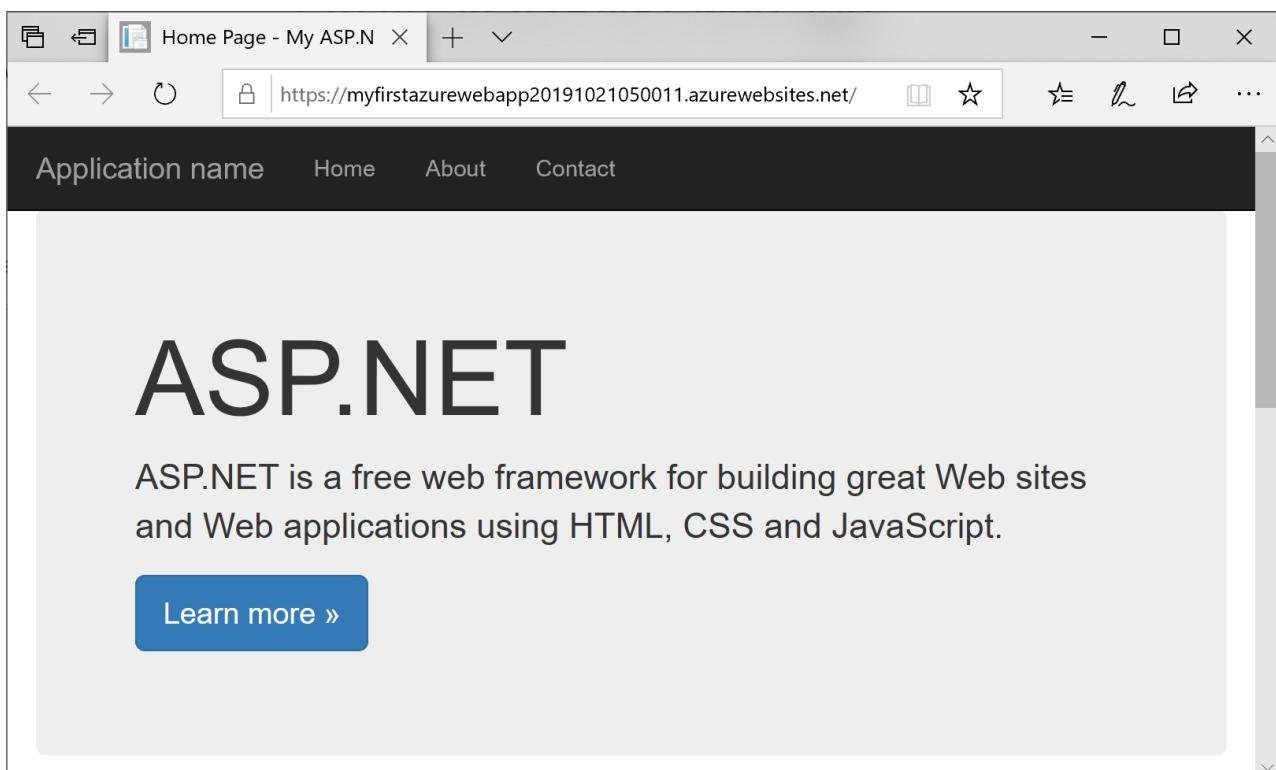
Hosting Plan

Application Insights

Explore additional Azure services
[Create a storage account](#)
[Create a SQL Database](#)

Clicking the Create button will create the following Azure resources
Hosting Plan - myAppServicePlan
App Service - myFirstAzureWebApp20191021042905

Once the wizard completes, it publishes the ASP.NET web app to Azure, and then launches the app in the default browser.



The app name specified in the **App Service Create new** page is used as the URL prefix in the format
`http://<app_name>.azurewebsites.net.`

Congratulations! Your ASP.NET web app is running live in Azure App Service.

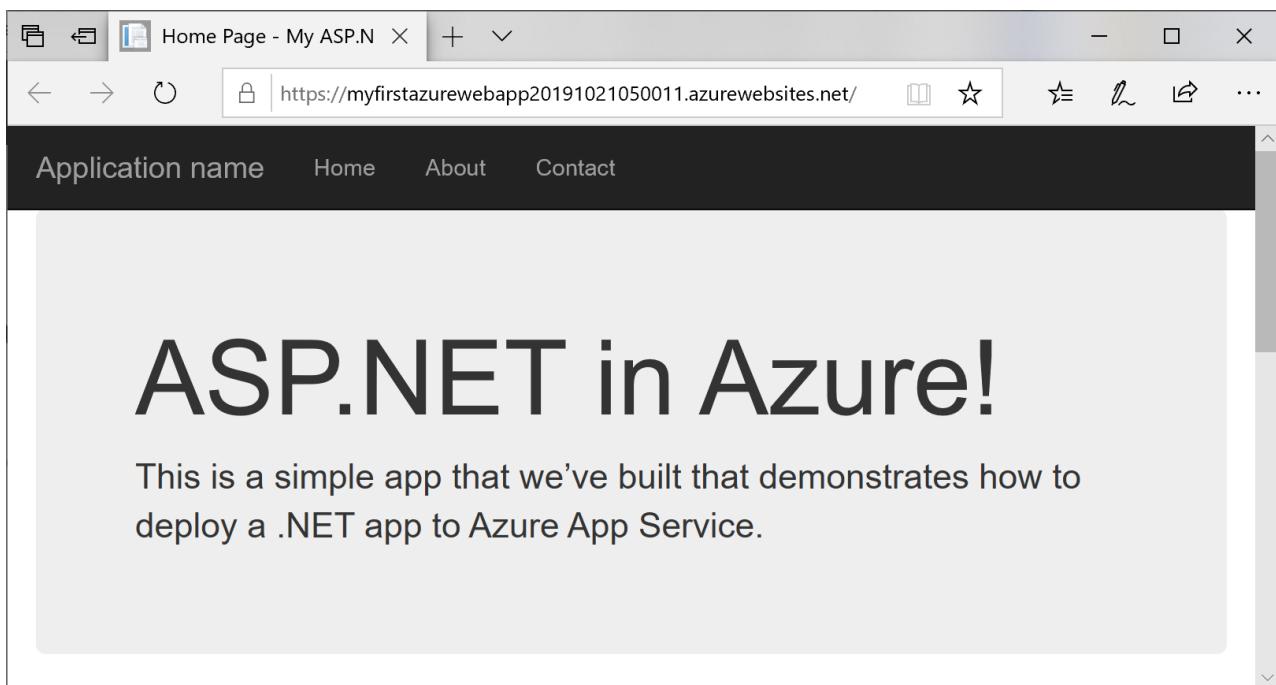
Update the app and redeploy

1. In **Solution Explorer**, under your project, open **Views > Home > Index.cshtml**.
2. Find the `<div class="jumbotron">` HTML tag near the top, and replace the entire element with the following code:

```
<div class="jumbotron">
    <h1>ASP.NET in Azure!</h1>
    <p class="lead">This is a simple app that we've built that demonstrates how to deploy a .NET app to
    Azure App Service.</p>
</div>
```

3. To redeploy to Azure, right-click the **myFirstAzureWebApp** project in **Solution Explorer** and select **Publish**. Then, select **Publish**.

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



Manage the Azure app

1. To manage the web app, go to the [Azure portal](#), and 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 "Azure services" section is visible, featuring a "Create a resource" button, an "App Services" icon, and links for "Azure Database for PostgreSQL" and "Azure Cosmos DB". To the right, a sidebar titled "Services" lists various Azure services, with "App Services" highlighted by a red box. Other listed services include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. Below the services list is a "Resources" section with a message stating "No results were found." At the bottom of the sidebar, there are navigation links for "Subscriptions", "Resource groups", and "All resources".

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

The screenshot shows the "App Services" blade in the Microsoft Azure portal. The title bar says "Home > App Services" and "App Services Microsoft". The top navigation bar includes "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", and "More". Below this, a message says "Subscriptions: All 2 selected - Don't see a subscription? Open Directory + Subscription settings". There are filter buttons for "Filter by na...", "All subsc...", "All resou...", "All locati...", "All tags", and "No group...". The main table displays "6 items" with columns: "Name" (checkbox), "Status", "App Type", and "App Service". The table lists four web apps: "cephalin320170403020701" (Running, Web App, test-sku), "denniseastusbot" (Running, Web App, z76-z763), "myFirstAzureWebApp2019..." (Running, Web App, ServiceP), and "WebApplicationASPDotNET..." (Running, Web App, ServicePl). The row for "myFirstAzureWebApp2019..." is highlighted with a red box.

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

The screenshot shows the Azure App Service Overview page for the application 'myFirstAzureWebApp20190826111115'. The left sidebar contains navigation links for Home, App Services, myFirstAzureWebApp20190826111115, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), and Settings (Configuration, Authentication / Authorization). The main content area displays the application's configuration details, including Resource group (myResourceGroup), Status (Running), Location (West Europe), Subscription (Contoso Subscription), Subscription ID (89d520cd-eb62-4f56-88a3-43087e91c288), and Tags (Click here to add tags). It also features three promotional cards: 'Diagnose and solve problems', 'Application Insights', and 'App Service Advisor'.

Home > App Services > myFirstAzureWebApp20190826111115

myFirstAzureWebApp20190826111115
App Service

Search (Ctrl+ /)

Browse Stop Swap Restart Delete Get publish profile Reset publish profile

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Deployment

Quickstart

Deployment slots

Deployment Center

Settings

Configuration

Authentication / Authorization

Resource group (change)
myResourceGroup

Status
Running

Location
West Europe

Subscription (change)
Contoso Subscription

Subscription ID
89d520cd-eb62-4f56-88a3-43087e91c288

Tags (change)
Click here to add tags

Click here to access Application Insights for monitoring and profiling for your ASP.NET Core app. →

URL
<https://myfirstazurewebapp20190826111115>

App Service Plan
myFirstAzureWebAppPlan (F1: Free)

FTP/deployment username
No FTP/deployment user set

FTP hostname
ftp:contoso-hosted-prod.ftp.azurewebsite.

FTPS hostname
ftps:contoso-hosted-prod.ftp.azurewebsite.

Diagnose and solve problems

Application Insights

App Service Advisor

The left menu provides different pages for configuring your app.

Next steps

[ASP.NET with SQL Database](#)

Create a Node.js web app in Azure

2/21/2020 • 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

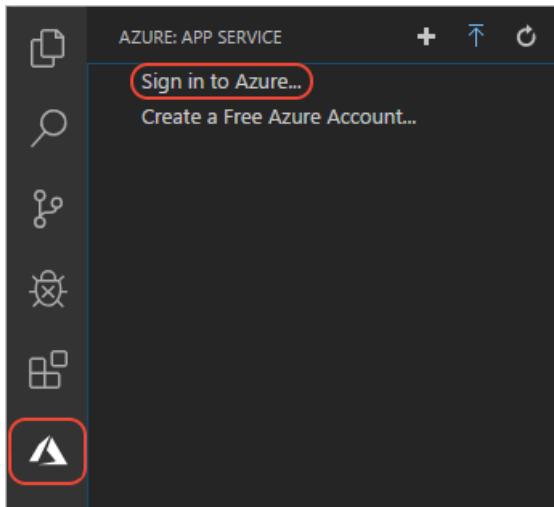
If you don't have an Azure account, [sign up today](#) for a free account with \$200 in Azure credits to try out any combination of services.

You need [Visual Studio Code](#) installed along with [Node.js](#) 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 on 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`.

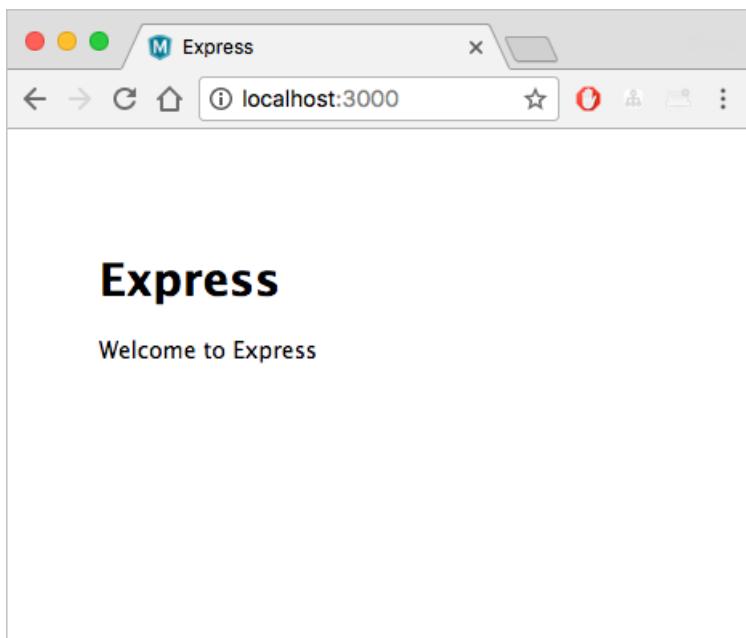
```
cd myExpressApp  
npm install
```

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

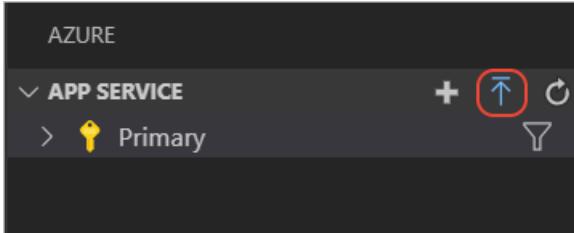
In this section, you deploy your Node.js app using VS Code and the Azure App Service extension. This quickstart uses the most basic deployment model where your app is zipped and deployed to an Azure Web App on Linux.

Deploy using Azure App Service

First, open your application folder in VS Code.

```
code .
```

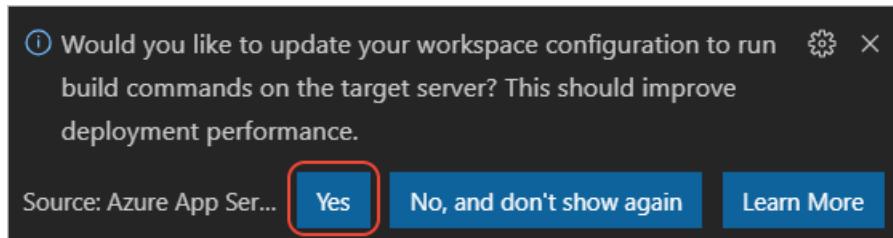
In the **AZURE APP SERVICE** explorer, select the blue up arrow icon to deploy your app to Azure.



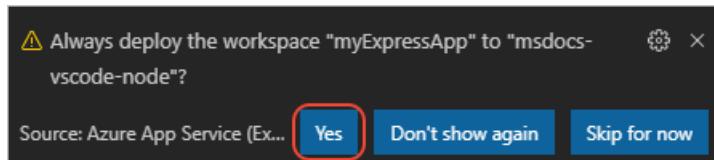
TIP

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 a creation option based on the operating system to which you want to deploy:
 - Linux: Choose **Create new Web App**.
 - Windows: Choose **Create new Web App** and select the **Advanced** option.
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. If targeting Linux, select a Node.js version when prompted. An **LTS** version is recommended.
5. If targeting Windows using the **Advanced** option, follow the additional prompts:
 - a. Select **Create a new resource group**, then enter a name for the resource group.
 - b. Select **Windows** for the operating system.
 - c. Select an existing App Service Plan or create a new one. You can select a pricing tier when creating a new plan.
 - d. Choose **Skip for now** when prompted about Application Insights.
 - e. Choose a region near you or near resources you wish to access.
6. After you respond to all the prompts, the notification channel shows the Azure resources that are being created for your app.
7. Select **Yes** when prompted to update your configuration to run `npm install` on the target server. Your app is then deployed.



8. 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

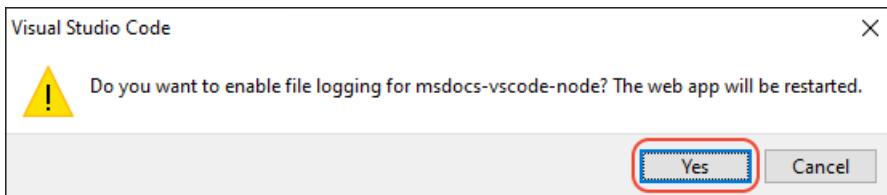
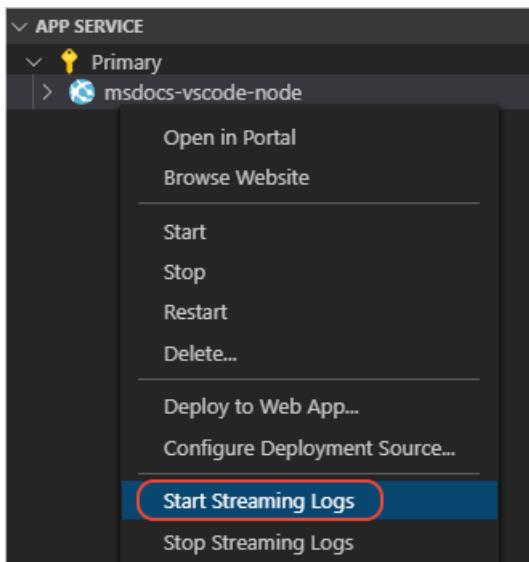
You can deploy changes to this app by using the same process and choosing the existing app rather than creating a new one.

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.

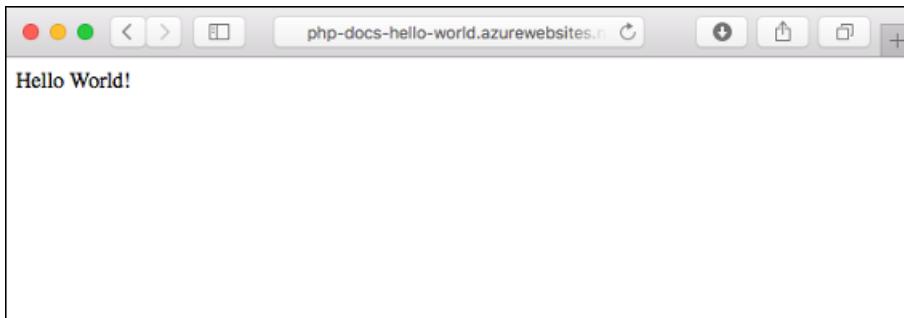
Create a PHP web app in Azure

2/20/2020 • 7 minutes to read • [Edit Online](#)

NOTE

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

[Azure App Service](#) provides a highly scalable, self-patching web hosting service. This quickstart tutorial shows how to deploy a PHP app to Azure App Service. You create the web app using the [Azure CLI](#) in Cloud Shell, and you use Git to deploy sample PHP code to the web app.



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

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

Prerequisites

To complete this quickstart:

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

Download the sample locally

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

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

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 Try It in the upper-right corner of a code block.
Selecting Try It doesn't automatically copy the code to Cloud Shell. | A screenshot of a code block. In the upper right corner, there is a green button labeled "Try It" with a white play icon. This button is highlighted with a red rectangle. |
| Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser. | A screenshot of the Azure portal's menu bar. It shows several icons: a blue square with a white triangle, a clipboard, a bell, a gear, and a question mark. The first icon (blue square with triangle) is highlighted with a red rectangle. |
| Select the Cloud Shell button on the menu bar at the upper right in the Azure portal . | A screenshot of the Azure portal's menu bar. It shows several icons: a blue square with a white triangle, a clipboard, a bell, a gear, and a question mark. The first icon (blue square with triangle) is highlighted with a red rectangle. |

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 in **Free** tier, run the `az appservice list-locations --sku FREE` command.

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

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

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

Create an Azure App Service plan

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

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

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

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": "app",
  "location": "West Europe",
  "maximumNumberOfWorkers": 1,
  "name": "myAppServicePlan",
  < JSON data removed for brevity. >
  "targetWorkerSizeId": 0,
  "type": "Microsoft.Web/serverfarms",
  "workerTierName": null
}
```

Create a web app

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

In the following example, replace `<app_name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `PHP|7.0`. To see all supported runtimes, run `az webapp list-runtimes`.

```
# 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",
  "enabled": true,
  < JSON data removed for brevity. >
}
```

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

NOTE

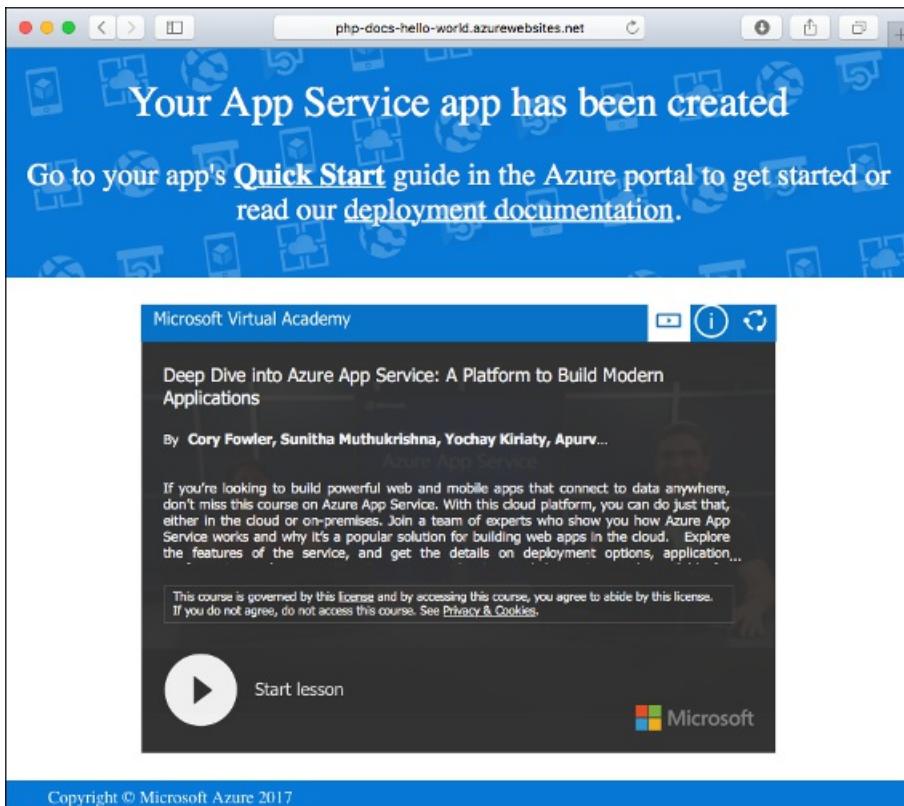
The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

```
https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git
```

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

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

Here is what your new web app should look like:



Push to Azure from Git

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:

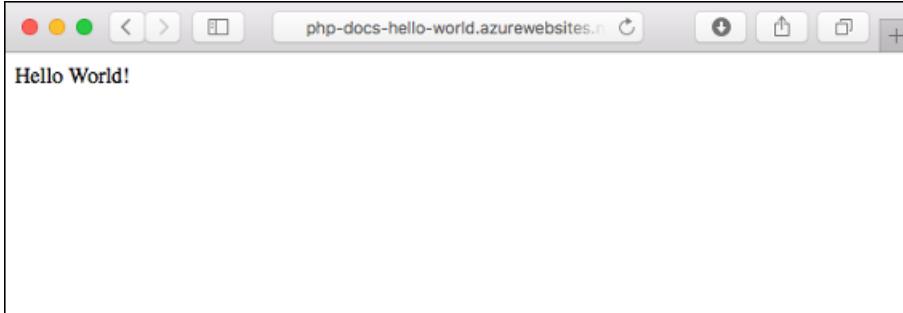
```
Counting objects: 2, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (2/2), 352 bytes | 0 bytes/s, done.  
Total 2 (delta 1), reused 0 (delta 0)  
remote: Updating branch 'master'.  
remote: Updating submodules.  
remote: Preparing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



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

Update locally and redeploy the code

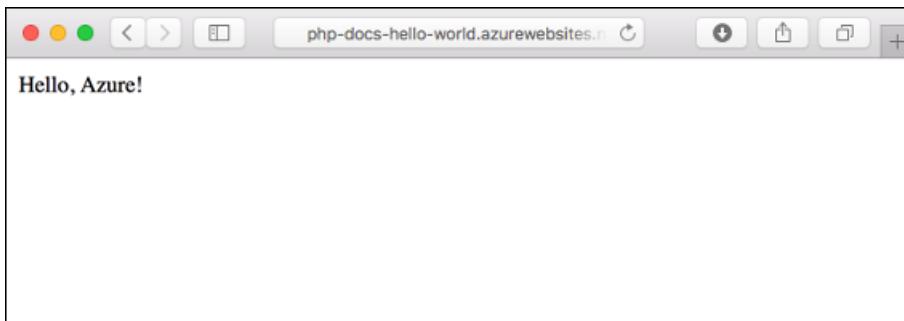
Using a local text editor, open the `index.php` file within the PHP app, and make a small change to the text within the string next to `echo`:

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

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

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

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



Manage your new Azure app

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

Services — All 58 results

- App Services**
- Function App
- Service Health
- App Service Certificates
- App Service Domains
- App Service Environments
- App Service plans
- Lab Services
- Media services
- Bot Services

2. Select the name of your Azure app.

Microsoft Azure

Home > App Services

App Services

| Name | Status | Location | Pricing Tier |
|----------------------|---------|-------------|--------------|
| php-docs-hello-world | Running | West Europe | Free |

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

The screenshot shows the Azure portal interface for a web application named "php-docs-hello-world". The main area displays various configuration details:

- Resource group (change): myResourceGroup
- Status: Running
- Location: West Europe
- Subscription (change): My Subscription Name
- Subscription ID: 00000000-0000-0000-000000000000
- URL: https://php-docs-hello-world.azurewebsites.net
- App Service Plan: myAppServicePlan (F1: Free)
- FTP/deployment username: No FTP/deployment user set
- FTP hostname: ftp://waws-prod-dm1-145.ftp.azurewebsites.windows.net
- FTPS hostname: https://waws-prod-dm1-145.ftp.azurewebsites.windows.net

Below the main details, there are two cards:

- Diagnose and solve problems**: A 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.

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

Clean up resources

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

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

This command may take a minute to run.

Next steps

[PHP with MySQL](#)

Quickstart: Create a Java app on Azure App Service on Windows

2/17/2020 • 3 minutes to read • [Edit Online](#)

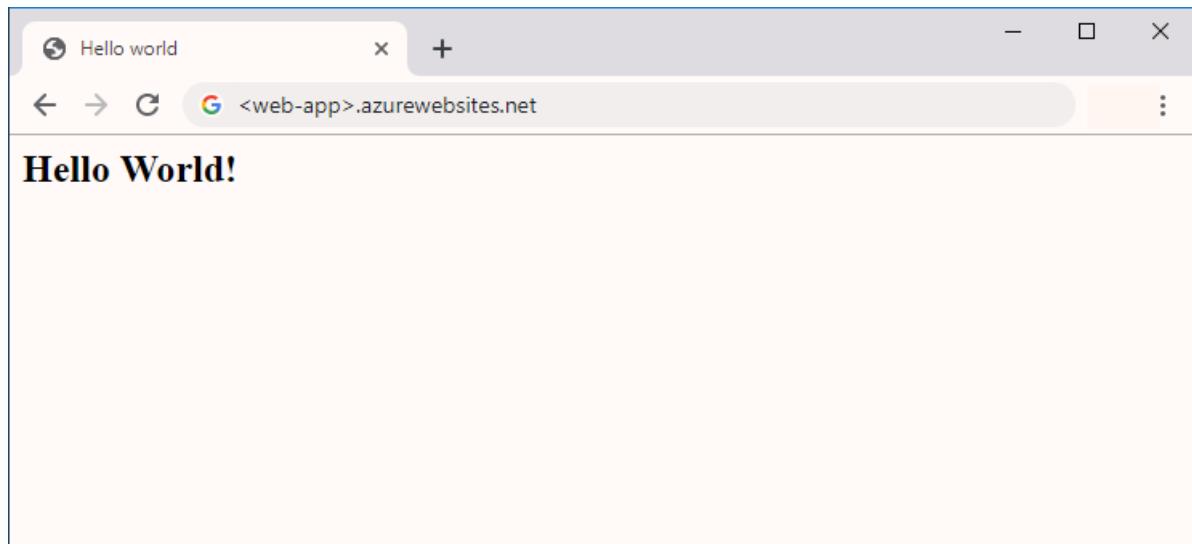
NOTE

This article deploys an app to App Service on Windows. To deploy to App Service on *Linux*, see [Create Java web app on Linux](#).

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

NOTE

The same thing can also be done using popular IDEs like IntelliJ and Eclipse. Check out our similar documents at [Azure Toolkit for IntelliJ Quickstart](#) or [Azure Toolkit for Eclipse 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 Try It in the upper-right corner of a code block.
Selecting Try It doesn't automatically copy the code to Cloud Shell. | Azure CLI |

| OPTION | EXAMPLE/LINK |
|---|--|
| Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser. |  |
| Select the Cloud Shell button on the menu bar at the upper right in the Azure portal . |  |

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

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

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
```

Configure the Maven plugin

To deploy from Maven, use the code editor in the Cloud Shell to open up the project `pom.xml` file in the `helloworld` directory.

```
code pom.xml
```

Then add the following plugin definition inside the `<build>` element of the `pom.xml` file.

```

<plugins>
    <!-- Deploy to Tomcat in App Service Windows -->
    <plugin>
        <groupId>com.microsoft.azure</groupId>
        <artifactId>azure-webapp-maven-plugin</artifactId>
        <version>1.8.0</version>
        <configuration>
            <!-- Specify v2 schema -->
            <schemaVersion>v2</schemaVersion>
            <!-- App information -->
            <subscriptionId>SUBSCRIPTION_ID</subscriptionId>
            <resourceGroup>RESOURCEGROUP_NAME</resourceGroup>
            <appName>WEBAPP_NAME</appName>
            <region>REGION</region>
            <!-- Java Runtime Stack for App Service on Windows-->
            <runtime>
                <os>windows</os>
                <javaVersion>1.8</javaVersion>
                <webContainer>tomcat 9.0</webContainer>
            </runtime>
            <deployment>
                <resources>
                    <resource>
                        <directory>${project.basedir}/target</directory>
                        <includes>
                            <include>*.war</include>
                        </includes>
                    </resource>
                </resources>
            </deployment>
        </configuration>
    </plugin>
</plugins>

```

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.

Update the following placeholders in the plugin configuration:

| PLACEHOLDER | DESCRIPTION |
|--------------------|--|
| SUBSCRIPTION_ID | The unique ID of the subscription you want to deploy your app to. Default subscription's ID can be found from the Cloud Shell or CLI using the <code>az account show</code> command. For all the available subscriptions, use the <code>az account list</code> command. |
| RESOURCEGROUP_NAME | Name for the new resource group in which to create your app. By putting all the resources for an app in a group, you can manage them together. For example, deleting the resource group would delete all resources associated with the app. Update this value with a unique new resource group name, for example, <i>myResourceGroup</i> . You will use this resource group name to clean up all Azure resources in a later section. |

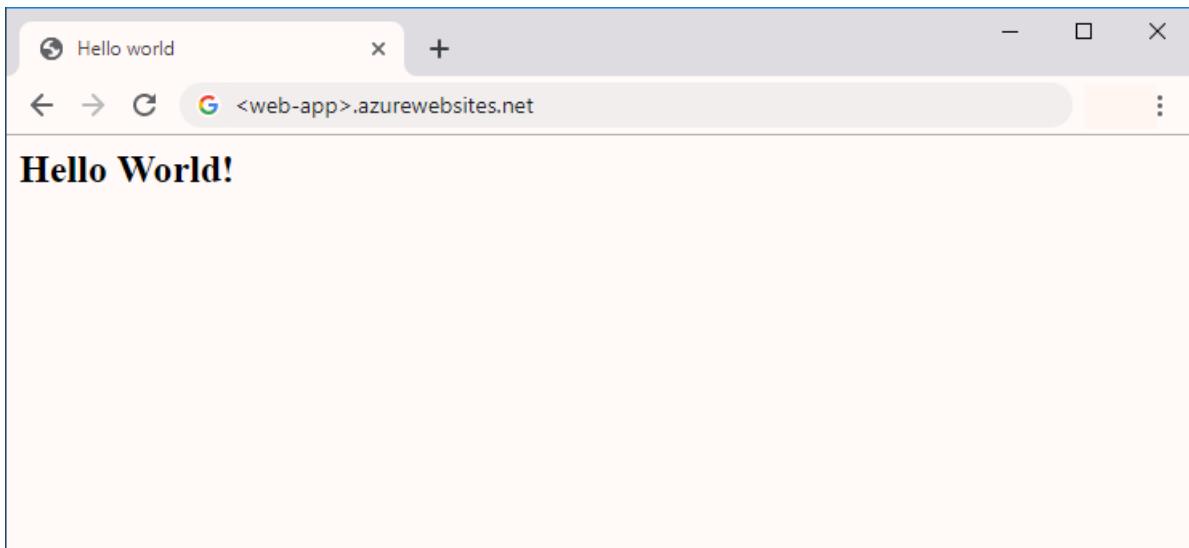
| PLACEHOLDER | DESCRIPTION |
|-------------|--|
| WEBAPP_NAME | The app name will be part of the host name for the app when deployed to Azure (WEBAPP_NAME.azurewebsites.net). Update this value with a unique name for the new App Service app, which will host your Java app, for example <i>contoso</i> . |
| REGION | An Azure region where the app is hosted, for example <i>westus2</i> . You can get a list of regions from the Cloud Shell or CLI using the <code>az account list-locations</code> command. |

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 Windows.

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

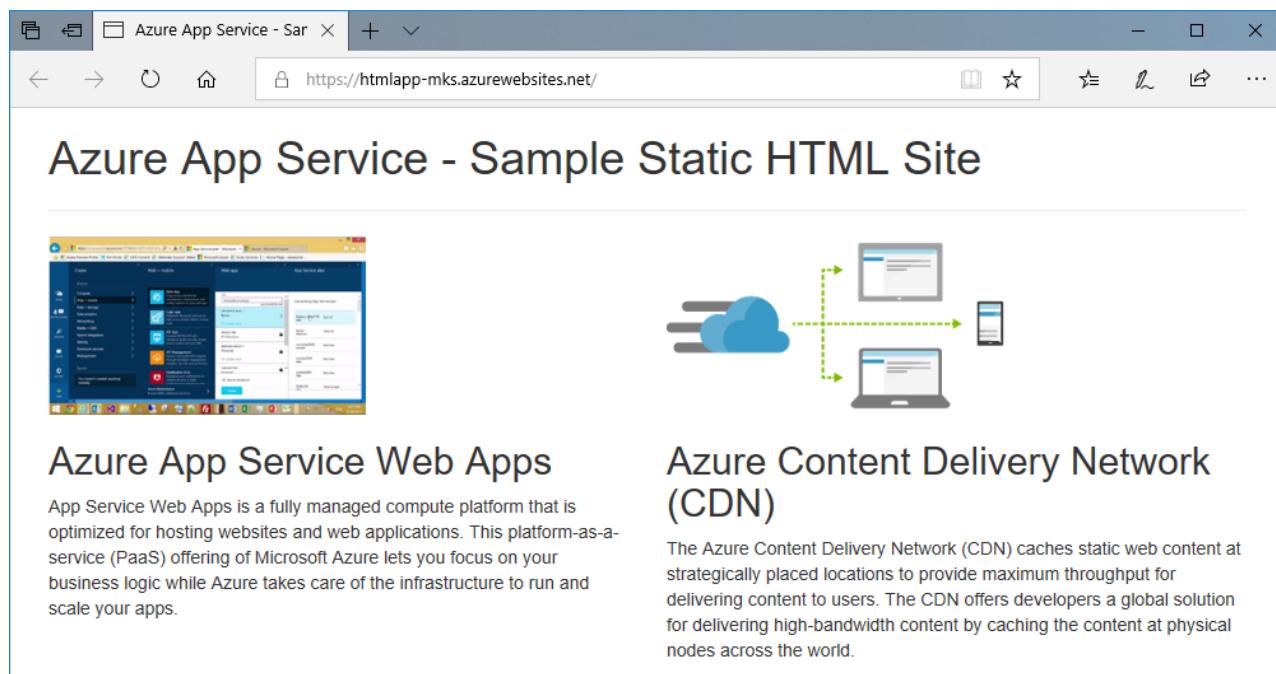
[Azure for Java Developers Resources](#)

[Map custom domain](#)

Create a static HTML web app in Azure

2/20/2020 • 3 minutes to read • [Edit Online](#)

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



Azure App Service Web Apps

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

Azure Content Delivery Network (CDN)

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

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

Use Azure Cloud Shell

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

To start Azure Cloud Shell:

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

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

1. Start Cloud Shell.

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

Download the sample

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

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

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

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

Create a web app

Change to the directory that contains the sample code and run the `az webapp up` command.

In the following example, replace <app_name> with a unique app name.

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

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

- Create a default resource group.
- Create a default app service plan.
- Create an app with the specified name.
- [Zip deploy](#) files from the current working directory to the web app.

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

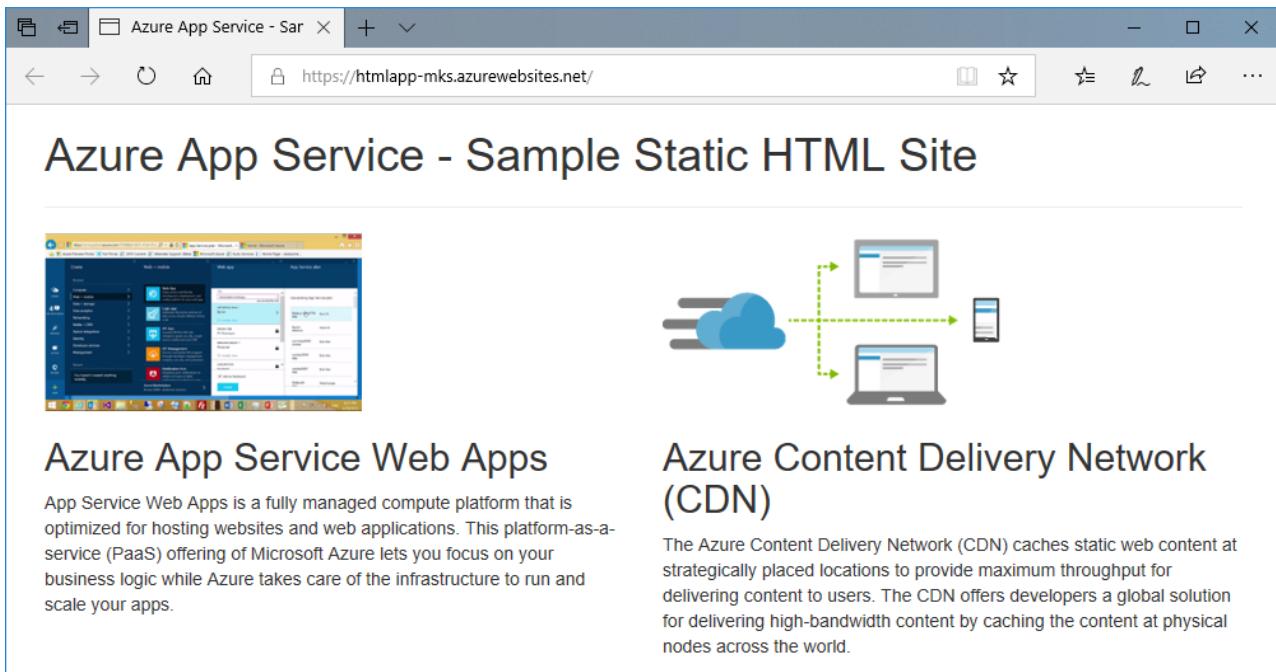
```
{  
  "app_url": "https://<app_name>.azurewebsites.net",  
  "location": "westeurope",  
  "name": "<app_name>",  
  "os": "Windows",  
  "resourcegroup": "appsvc_rg_Windows_westeurope",  
  "serverfarm": "appsvc_asp_Windows_westeurope",  
  "sku": "FREE",  
  "src_path": "/home/<username>/quickstart/html-docs-hello-world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

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

Browse to the app

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

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

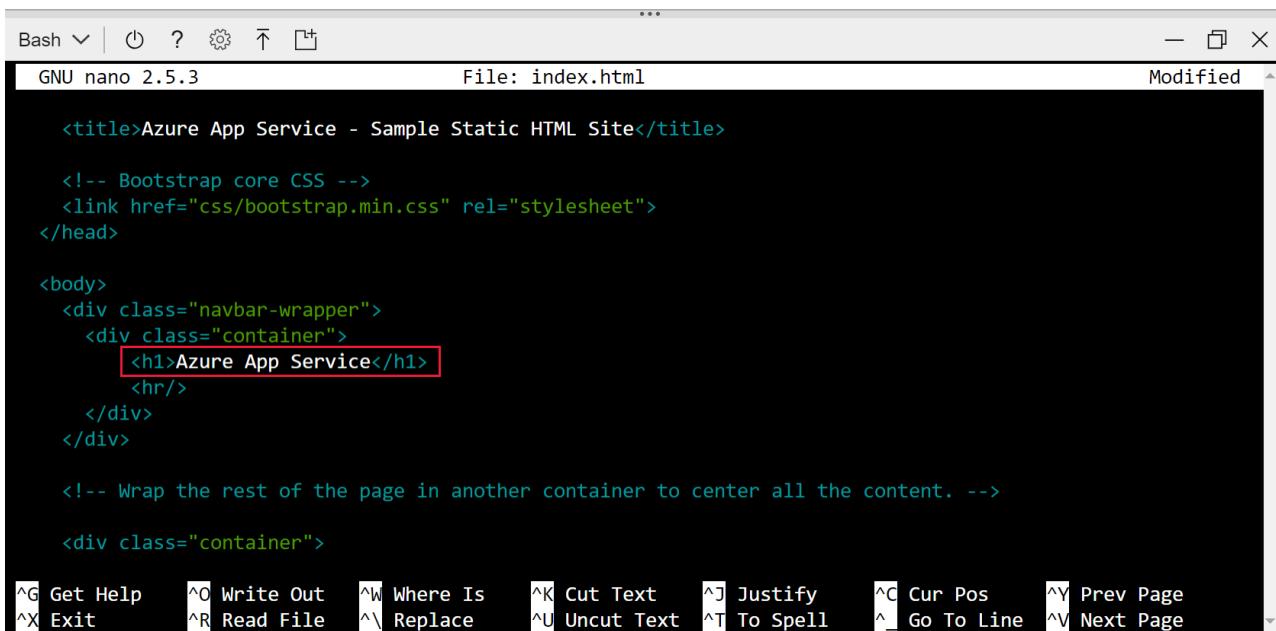


A screenshot of a Microsoft Edge browser window titled "Azure App Service - Sar". The address bar shows the URL `https://htmlapp-mks.azurewebsites.net/`. The main content area displays the title "Azure App Service - Sample Static HTML Site". Below the title, there is a section titled "Azure App Service Web Apps" with a brief description and a screenshot of the Azure portal. To the right of this section is a diagram illustrating the Azure Content Delivery Network (CDN) architecture, showing a cloud icon connected to multiple devices (laptop, smartphone, tablet) via dashed green arrows. Another section titled "Azure Content Delivery Network (CDN)" provides more details about its function.

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

Update and redeploy the app

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



A screenshot of the Cloud Shell terminal showing the nano text editor. The terminal title is "Bash". The command `File: index.html` is displayed above the editor area. The editor content shows the HTML code for the static website. A red box highlights the `<h1>Azure App Service</h1>` line. The bottom of the terminal shows the nano key bindings.

```
GNU nano 2.5.3 File: index.html Modified ^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^V Next Page
```

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

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

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

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

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

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

Manage your new Azure app

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

Services

- App Services
- Function App
- Service Health
- App Service Certificates
- App Service Domains
- App Service Environments
- App Service plans
- Lab Services
- Media services
- Bot Services

Recent resources

| NAME |
|-------------------------|
| cephalin320170403020701 |

Navigate

- Subscriptions
- Resource groups
- All resources

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

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

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

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

htmlapp-mks App Service

Resource group (change) : azscreens URL : <https://htmlapp-mks.azurewebsites.net>

Status : Running App Service Plan : test-sku (B1: 1)

Location : Central US FTP/deployment usernam... : htmlapp-mks\Deploy1234567

Subscription (change) : A Subscription FTP hostname : <ftp://waws-prod-dm1-145.ftp.azurewebsites.windows....>

Subscription ID : 12345678-1234-1234-1234-123456781234 FTPS hostname : <ftps://waws-prod-dm1-145.ftp.azurewebsites.windows....>

Tags (change) : Click here to add tags

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

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

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

Http 5xx
8:15 AM 8:30 AM 8:45 AM 9 AM
Http Server Errors (Sum) donetappsqldb1234x
0

Data In
8:15 AM 8:30 AM 8:45 AM 9 AM
Data In (Sum) donetappsqldb1234x
8.28 kB

Data Out
8:15 AM 8:30 AM 8:45 AM 9 AM
Data Out (Sum) donetappsqldb1234x
14.28 kB

The left menu provides different pages for configuring your app.

Clean up resources

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

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

This command may take a minute to run.

Next steps

[Map custom domain](#)

Run a custom Windows container in Azure (Preview)

1/13/2020 • 4 minutes to read • [Edit Online](#)

Azure App Service provides pre-defined application stacks on Windows like ASP.NET or Node.js, running on IIS. The preconfigured Windows environment locks down the operating system from administrative access, software installations, changes to the global assembly cache, and so on. For more information, see [Operating system functionality on Azure App Service](#). If your application requires more access than the preconfigured environment allows, you can deploy a custom Windows container instead.

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

Prerequisites

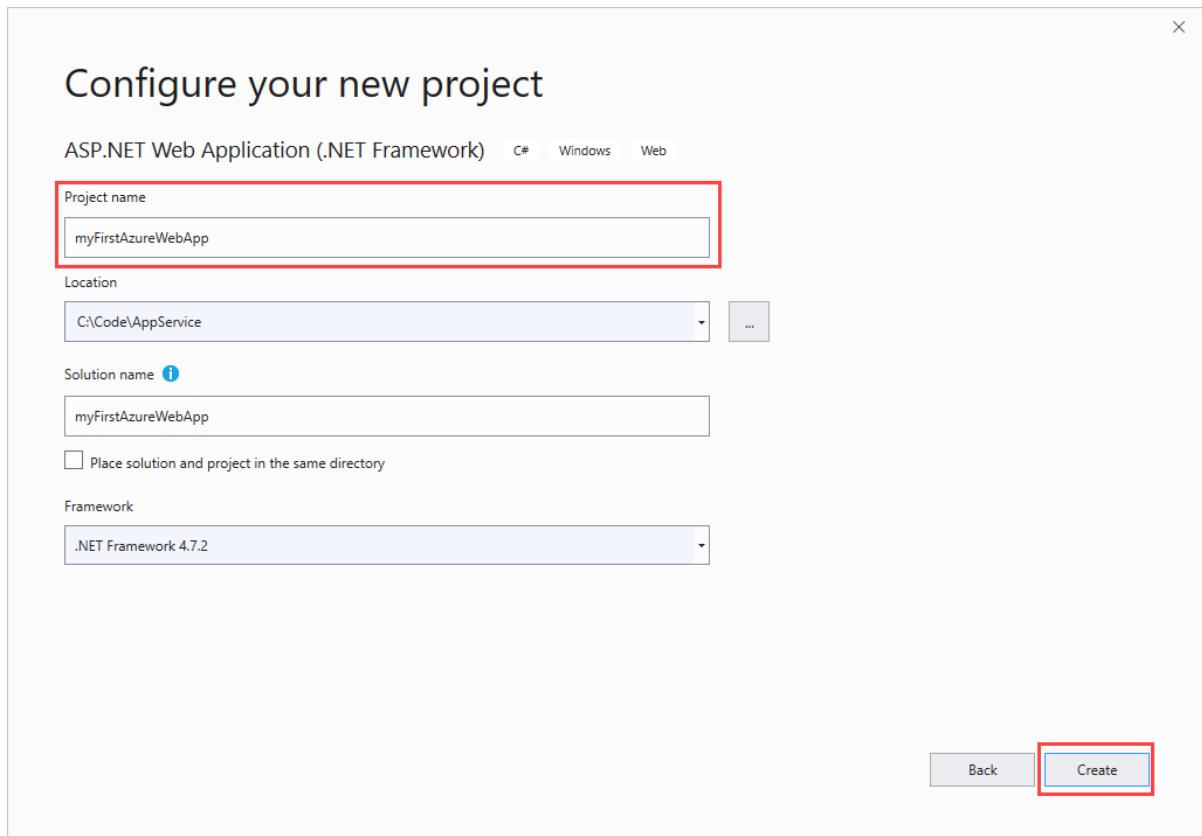
To complete this tutorial:

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

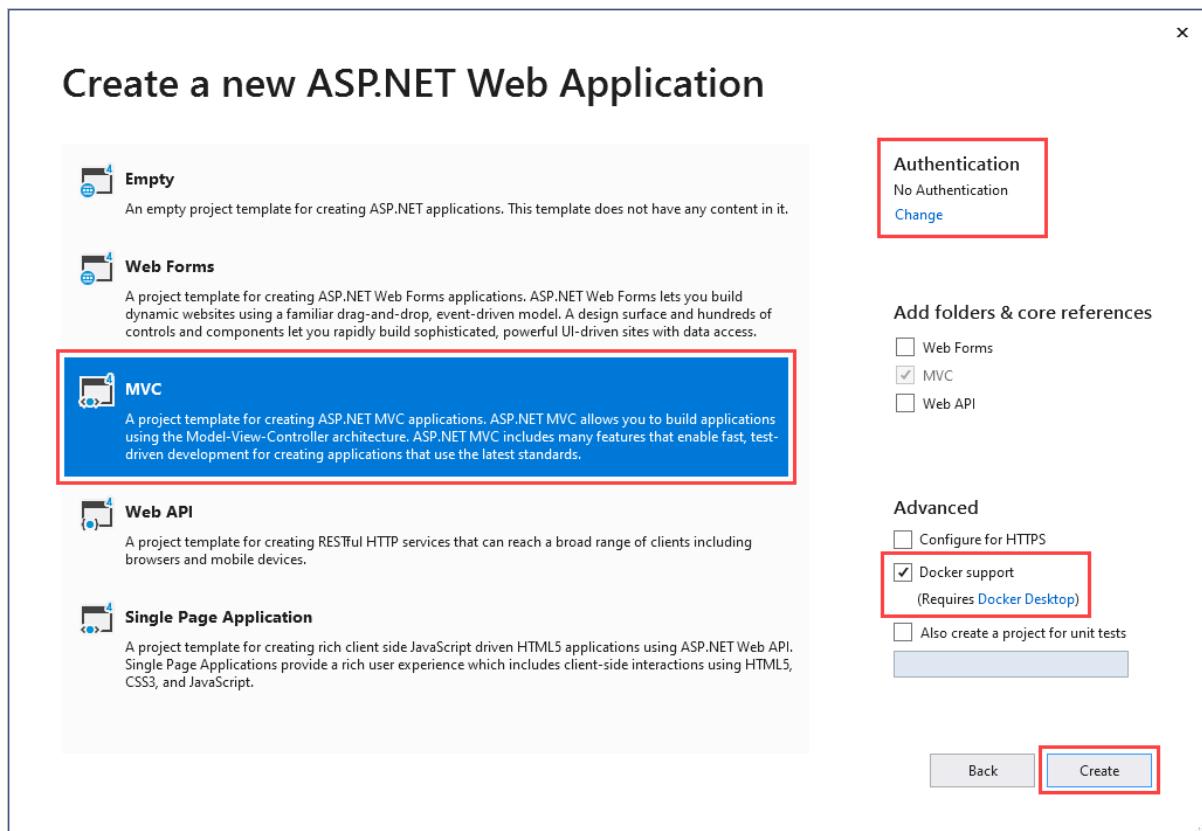
Create an ASP.NET web app

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

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



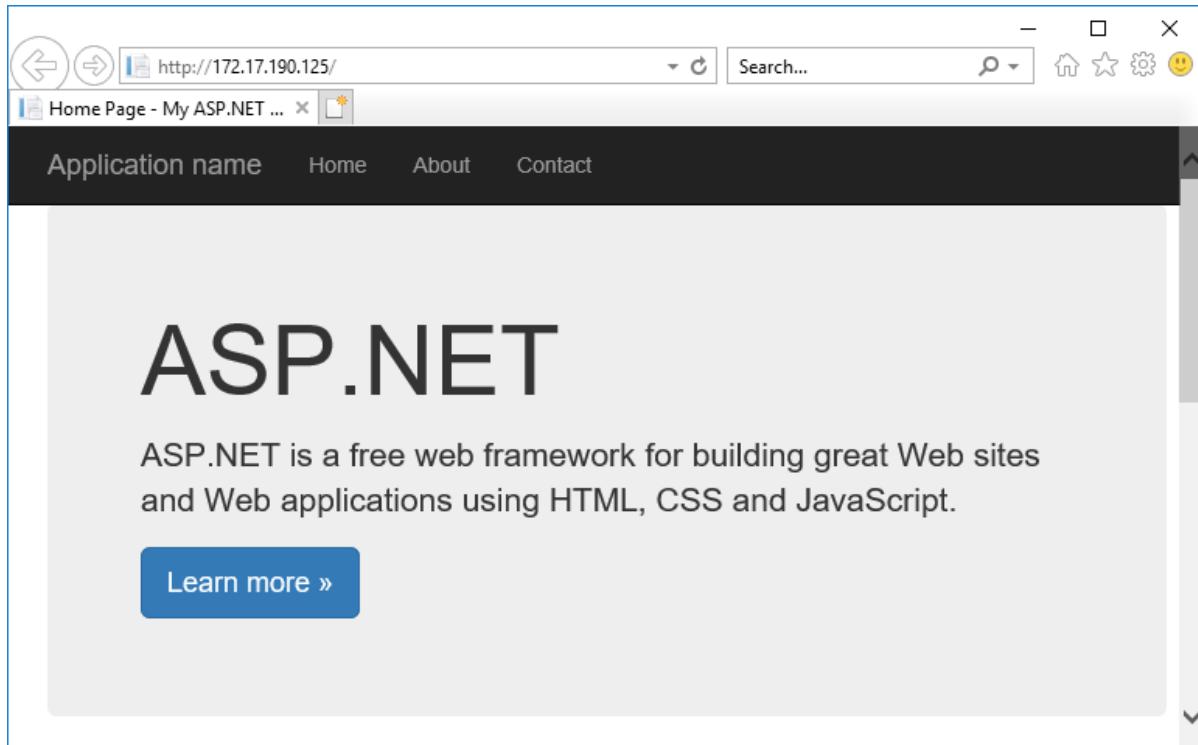
4. You can deploy any type of ASP.NET web app to Azure. For this quickstart, choose the **MVC** template.
5. Select **Docker support**, and make sure authentication is set to **No Authentication**. Select **Create**.



6. If the *Dockerfile* file isn't opened automatically, open it from the **Solution Explorer**.
7. You need a **supported parent image**. Change the parent image by replacing the `FROM` line with the following code and save the file:

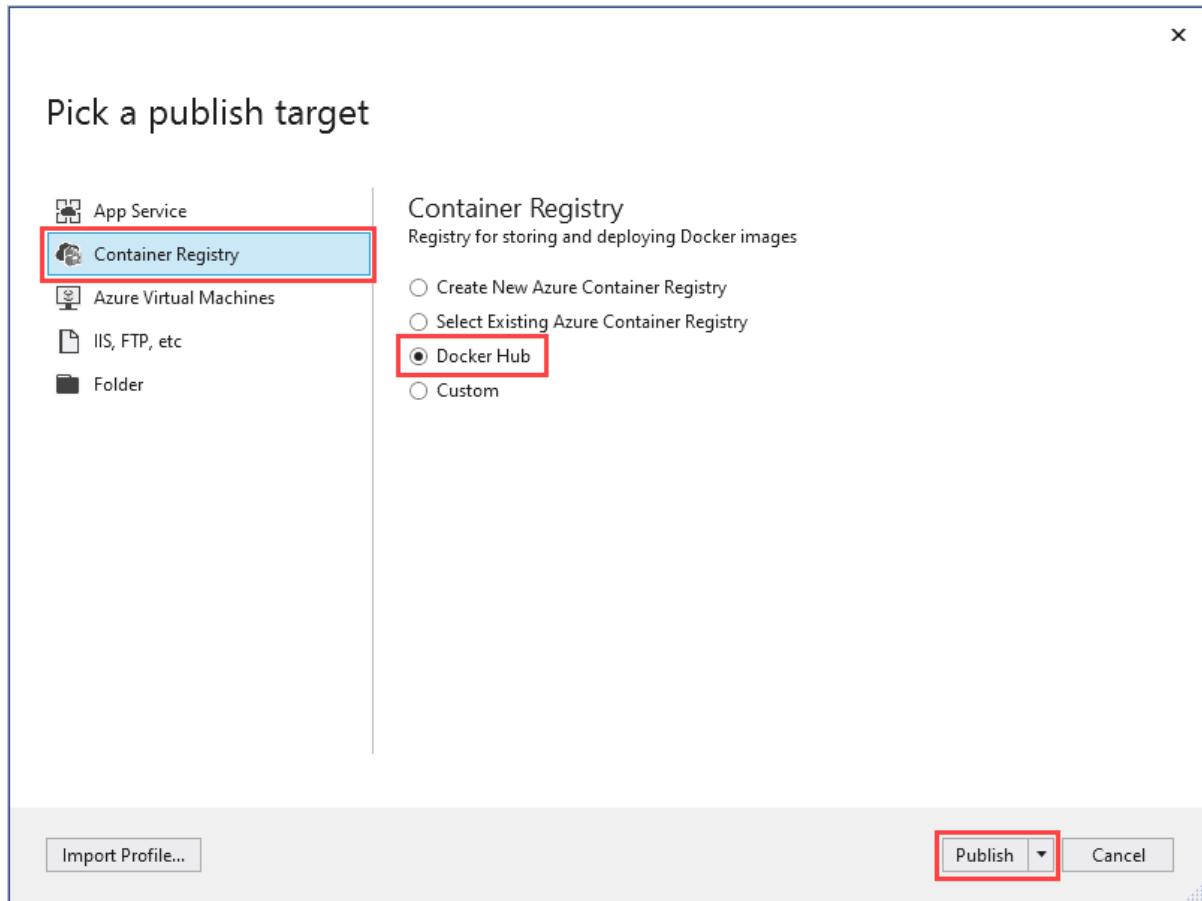
```
FROM mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019
```

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



Publish to Docker Hub

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



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

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

myFirstAzureWebApp

Overview Publish

Connected Services Deploy your app to a folder, IIS, Azure, or another destination. [More info](#)

Publish

registry.hub.docker.com/_contoso

New Edit Rename Delete

Summary

Actions

Image Tag: latest

Repository: https://registry.hub.docker.com/_contoso

Configuration: Release

Edit Image Tag

5. Copy this repository name for later.

Create a Windows container app

1. Sign in to the [Azure portal](#).
2. Choose **Create a resource** in the upper left-hand corner of the Azure portal.
3. In the search box above the list of Azure Marketplace resources, search for **Web App for Containers**, and select **Create**.
4. In **Web App Create**, choose your subscription and a **Resource Group**. You can create a new resource

group if needed.

5. Provide an app name, such as *win-container-demo* and choose **Windows** for **Operating System**. Select **Next: Docker** to continue.

The screenshot shows the Microsoft Azure portal interface for creating a new Web App. The left sidebar lists various services under 'FAVORITES'. The main form is titled 'Web App' and includes sections for 'Instance Details', 'App Service Plan', and 'Sku and size'. The 'Subscription' dropdown is set to 'Contoso Subscription' and the 'Resource Group' dropdown is set to 'AppServiceRG'. In the 'Instance Details' section, the 'Name' field is filled with 'win-container-demo'. The 'Operating System' dropdown has 'Windows' selected. In the 'App Service Plan' section, the 'Windows Plan (West US)' dropdown shows '(New) ASP-AppServiceRG'. The 'Sku and size' dropdown shows 'Premium Container PC2'. At the bottom, there are 'Review and create' and 'Next: Docker >' buttons, with 'Next: Docker >' being highlighted by a red box.

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

Web App

Basics Docker Monitoring Tags Review + create

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

Image Source

Docker Hub

Docker hub options

Access Type *

Public

Image and tag *

contoso/myFirstAzureWebApp:latest

Review + create

< Previous

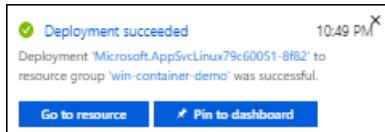
Next : Monitoring >

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

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

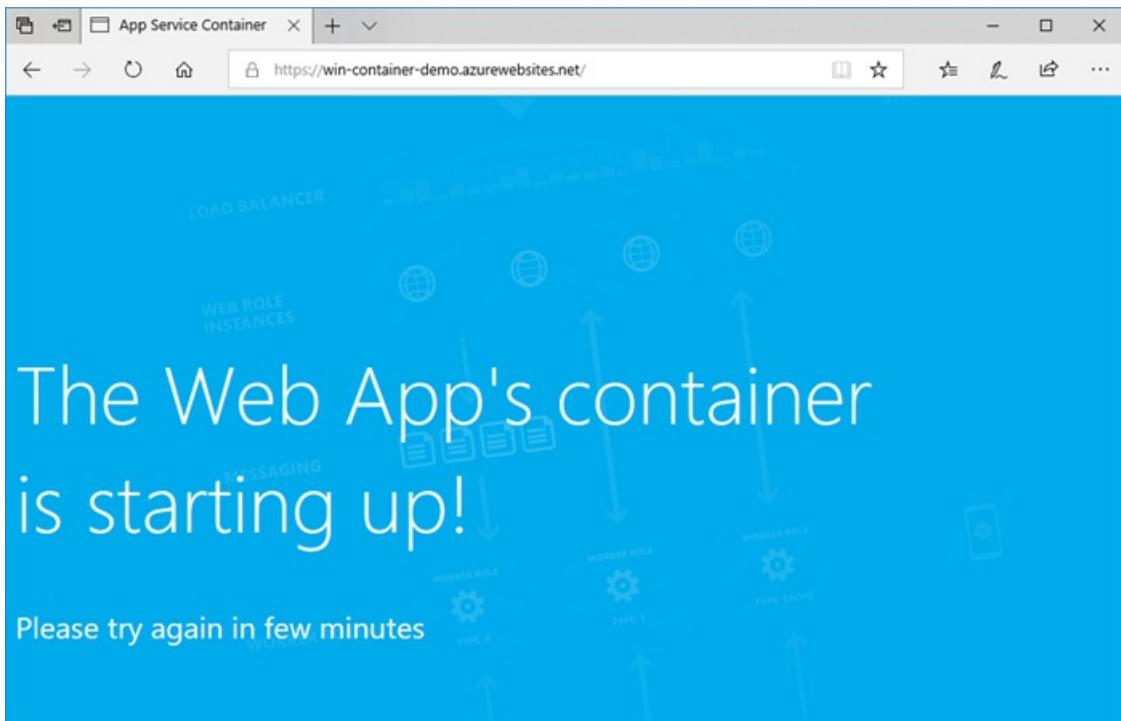
Browse to the container app

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

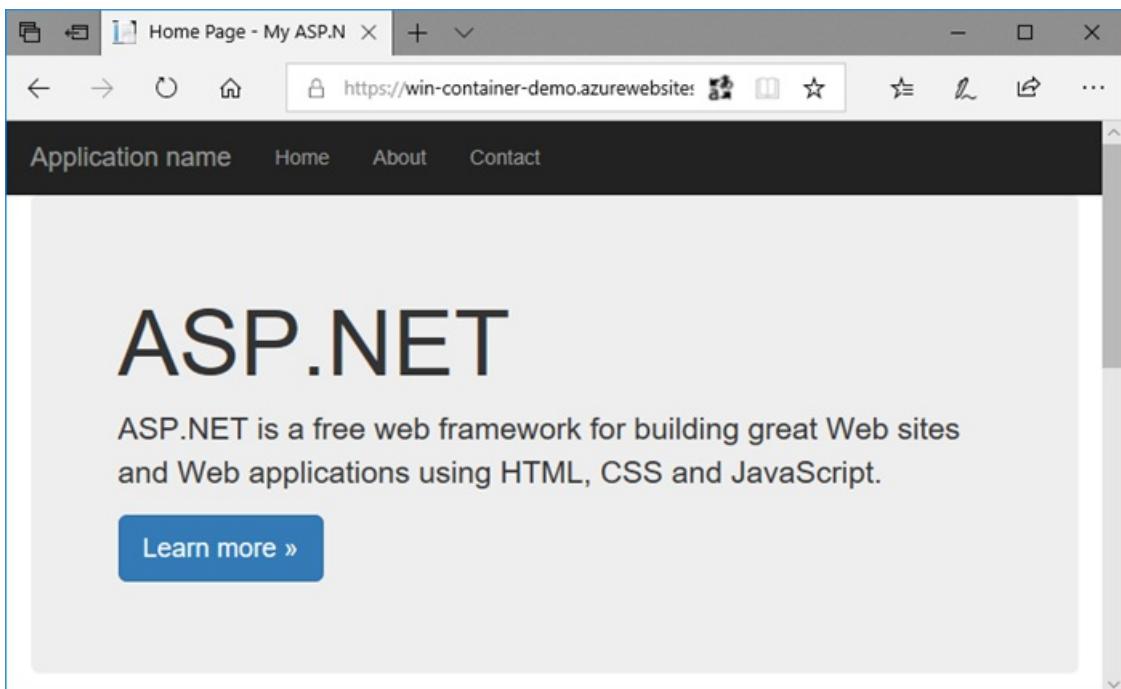


1. Click **Go to resource**.
2. In the overview of this resource, follow the link next to **URL**.

A new browser page opens to the following page:



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



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

See container start-up logs

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

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

The streamed logs looks like this:

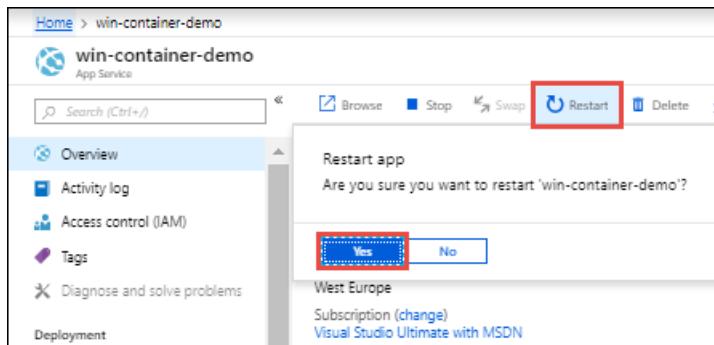
```
2018-07-27T12:03:11 Welcome, you are now connected to log-streaming service.  
27/07/2018 12:04:10.978 INFO - Site: win-container-demo - Start container succeeded. Container:  
facbf6cb214de86e58557a6d073396f640bbe2fdec88f8368695c8d1331fc94b  
27/07/2018 12:04:16.767 INFO - Site: win-container-demo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

Update locally and redeploy

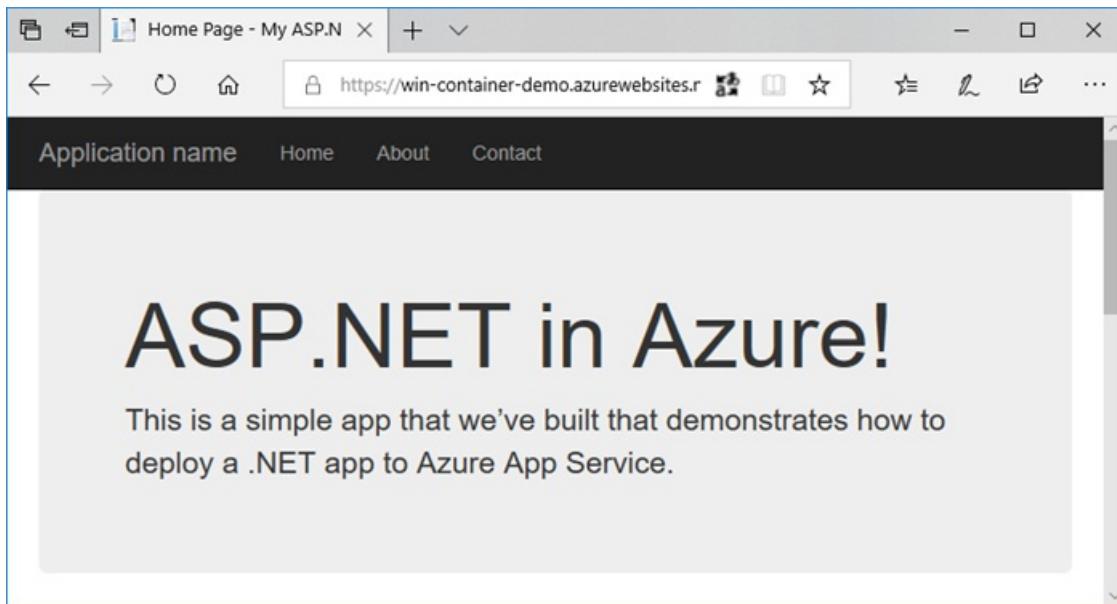
1. In Visual Studio, in **Solution Explorer**, open **Views > Home > Index.cshtml**.
2. Find the `<div class="jumbotron">` HTML tag near the top, and replace the entire element with the following code:

```
<div class="jumbotron">  
    <h1>ASP.NET in Azure!</h1>  
    <p class="lead">This is a simple app that we've built that demonstrates how to deploy a .NET app to  
    Azure App Service.</p>  
</div>
```

3. To redeploy to Azure, right-click the **myFirstAzureWebApp** project in **Solution Explorer** and choose **Publish**.
4. On the publish page, select **Publish** and wait for publishing to complete.
5. To tell App Service to pull in the new image from Docker Hub, restart the app. Back in the app page in the portal, click **Restart > Yes**.



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



Use a different parent image

You're free to use a different custom Docker image to run your app. However, you must choose the right [parent image \(base image\)](#) for the framework you want:

- To deploy .NET Framework apps, use a parent image based on the Windows Server Core 2019 [Long-Term Servicing Channel \(LTSC\)](#) release.
- To deploy .NET Core apps, use a parent image based on the Windows Server Nano 1809 [Semi-Annual Servicing Channel \(SAC\)](#) release.

It takes some time to download a parent image during app start-up. However, you can reduce start-up time by using one of the following parent images that are already cached in Azure App Service:

- mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019
- mcr.microsoft.com/windows/nanoserver:1809 - this image is the base container used across Microsoft [ASP.NET Core](#) Microsoft Windows Nano Server images.

Next steps

[Migrate to Windows container in Azure](#)

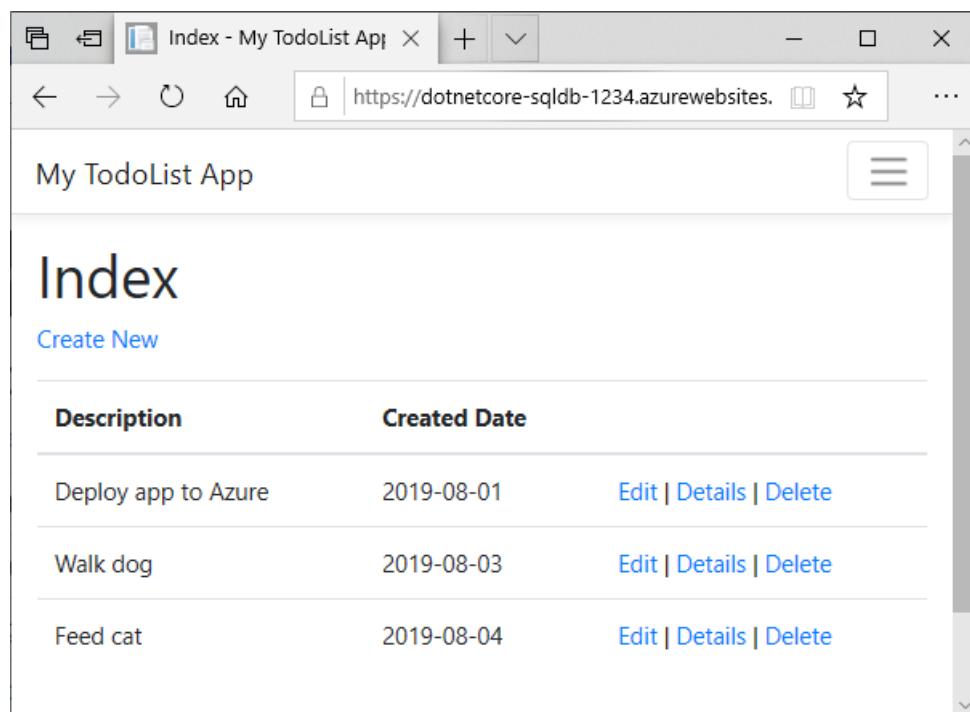
Tutorial: Build an ASP.NET Core and SQL Database app in Azure App Service

2/20/2020 • 13 minutes to read • [Edit Online](#)

NOTE

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

[App Service](#) provides a highly scalable, self-patching web hosting service in Azure. 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.



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

What 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](#)

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

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

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

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 in **Free** tier, run the `az appservice list-locations --sku FREE` command.

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

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

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

Create a SQL Database logical server

In the Cloud Shell, create a SQL Database logical server with the `az sql server create` command.

Replace the *<server_name>* placeholder with a unique SQL Database name. This name is used as the part of the 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 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](#).

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.

Configure local git deployment

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

To configure the deployment user, run the `az webapp deployment user set` command in Azure Cloud Shell.

Replace <username> and <password> with a deployment user username and password.

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

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

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

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

Create an App Service plan

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

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

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

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

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

Create a web app

Create a **web app** in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --deployment-local-git
```

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

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

NOTE

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

```
https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git
```

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<MyDatabaseContext>(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<MyDatabaseContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("MyDbConnection")));
else
    services.AddDbContext<MyDatabaseContext>(options =>
        options.UseSqlite("Data Source=localdatabase.db"));

// Automatically perform database migration
services.BuildServiceProvider().GetService<MyDatabaseContext>().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.

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.

IMPORTANT

For production apps that need to scale out, follow the best practices in [Applying migrations in production](#).

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.

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([Bind("ID,Description,CreatedDate")] Todo todo)` method and add `Done` to the list of properties in the `Bind` attribute. When you're done, your `Create()` method signature looks like the following code:

```
public async Task<IActionResult> Create([Bind("ID,Description,CreatedDate,Done")] Todo todo)
```

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.

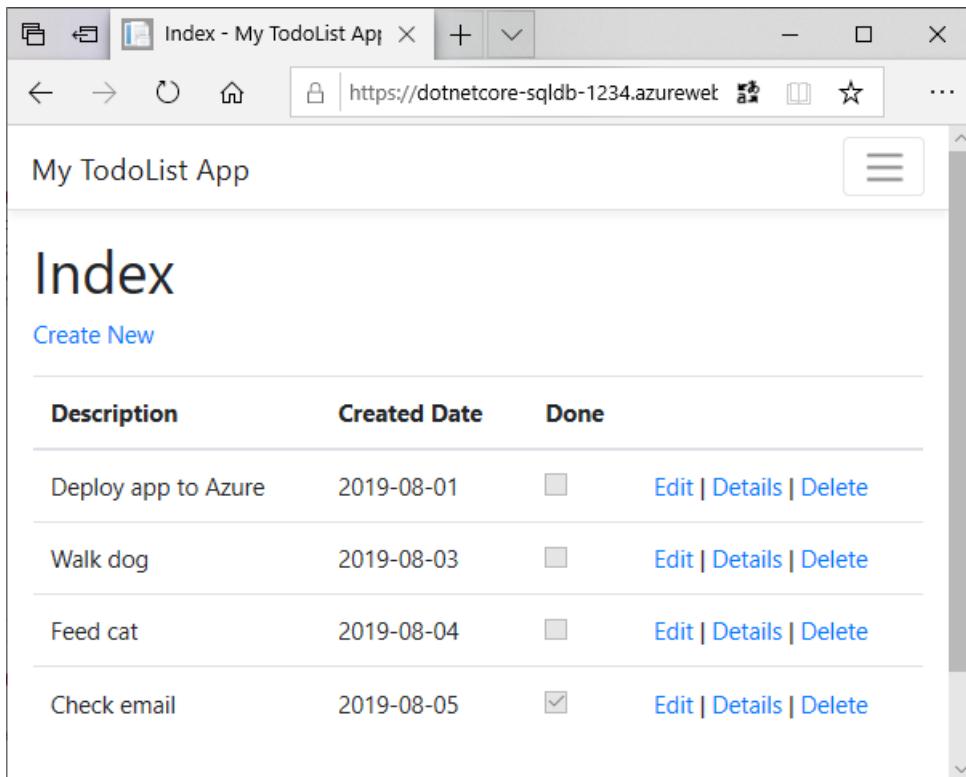
```
dotnet run
```

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 App Service app and try adding a to-do item and check **Done**.



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

All your existing to-do items are still displayed. When you republish your .NET Core app, existing data in your SQL Database isn't lost. Also, Entity Framework Core Migrations only changes the data schema and leaves your existing data intact.

Stream diagnostic logs

While the ASP.NET Core app runs in Azure App Service, you can get the console logs piped to the Cloud Shell. That way, you can get the same diagnostic messages to help you debug application errors.

The sample project already follows the guidance 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 `Program.cs`.

To set the ASP.NET Core [log level](#) in App Service to [Information](#) from the default level [Error](#), use the [az webapp log config](#) command in the Cloud Shell.

```
az webapp log config --name <app_name> --resource-group myResourceGroup --application-logging true --level information
```

NOTE

The project's log level is already set to [Information](#) in *appsettings.json*.

To start log streaming, use the [az webapp log tail](#) command in the Cloud Shell.

```
az webapp log tail --name <app_name> --resource-group myResourceGroup
```

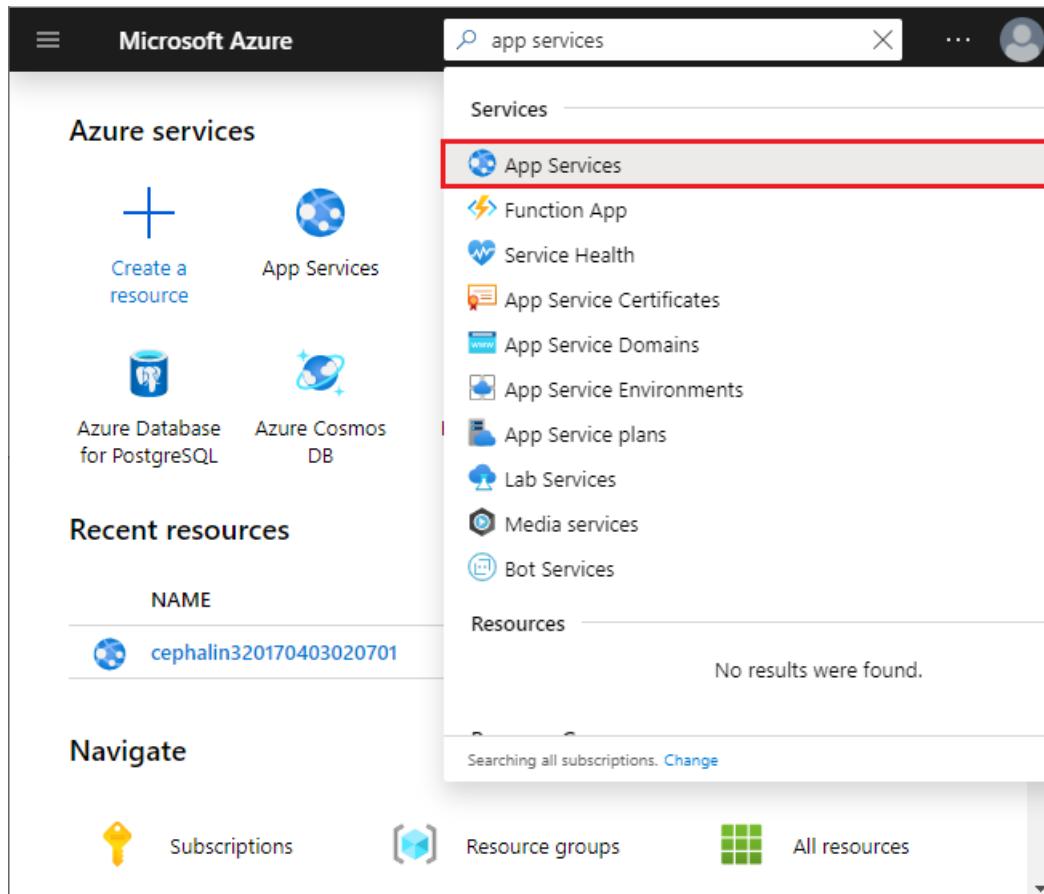
Once log streaming has started, refresh the Azure app in the browser to get some web traffic. You can now see console logs piped to the terminal. If you don't see console logs immediately, check again in 30 seconds.

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

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



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

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

Name	Status	App Type	App Service Plan	Location
cephalin320170403020701	Running	Web App	test-sku	Central US
denniseastusbot	Running	Web App	z76-z763if-sp	East US
DotNetAppSqlDb1234	Running	Web App	ServicePlane917530d...	Central US
myFirstAzureWebApp20190...	Running	Web App	ServicePlane917530d...	Central US
myfunctionapp04	Running	Function App	ASP-TimRG01-a65e	East US 2

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 left side of the page shows the different configuration pages you can open.

DotNetAppSqlDb1234 App Service

Resource group (change) : azscreens

Status : Running

Location : Central US

Subscription (change) : A Subscription

Subscription ID : 12345678-1234-1234-1234-123456781234

Tags (change) : Click here to add tags

URL : <https://dotnetappsqldb1234.azurewebsites.net>

App Service Plan : ServicePlane917530d-821f (S1: 1)

FTP/deployment userna... : DotNetAppSqlDb1234(Deploy1234567)

FTP hostname : <ftp://waws-prod-dm1-087.ftp.azurewebsites.windows.net>

FTPS hostname : <ftps://waws-prod-dm1-087.ftp.azurewebsites.windows....>

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

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

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

Http 5xx

Data In

Data Out

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

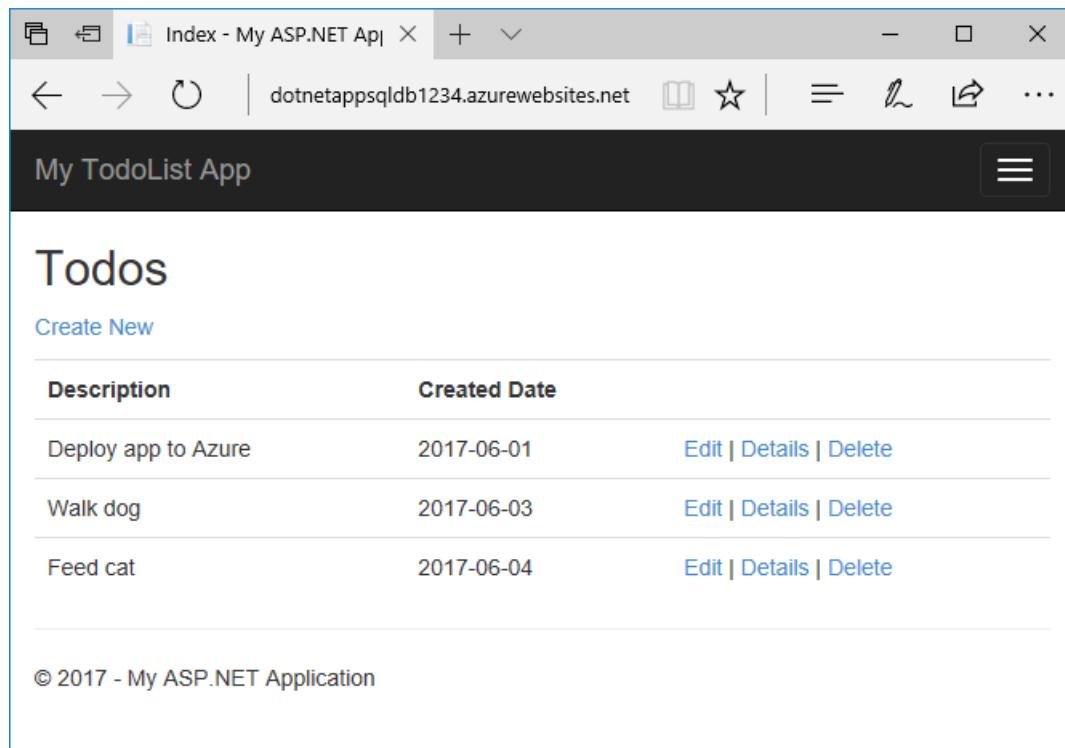
Advance to the next tutorial to learn how to map a custom DNS name to your app.

[Map an existing custom DNS name to Azure App Service](#)

Tutorial: Build an ASP.NET app in Azure with SQL Database

12/2/2019 • 11 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows you how to deploy a data-driven ASP.NET app in App Service and connect it to [Azure SQL Database](#). When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database.



In this tutorial, you learn how to:

- Create a SQL Database in Azure
- Connect an ASP.NET app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

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

Prerequisites

To complete this tutorial:

Install [Visual Studio 2019](#) with the **ASP.NET and web development** workload.

If you've installed Visual Studio already, add the workloads in Visual Studio by clicking **Tools > Get Tools and Features**.

Download the sample

- Download the sample project.
- Extract (unzip) the *dotnet-sqldb-tutorial-master.zip* file.

The sample project contains a basic **ASP.NET MVC** create-read-update-delete (CRUD) app using **Entity Framework Code First**.

Run the app

Open the *dotnet-sqldb-tutorial-master/DotNetAppSqlDb.sln* file in Visual Studio.

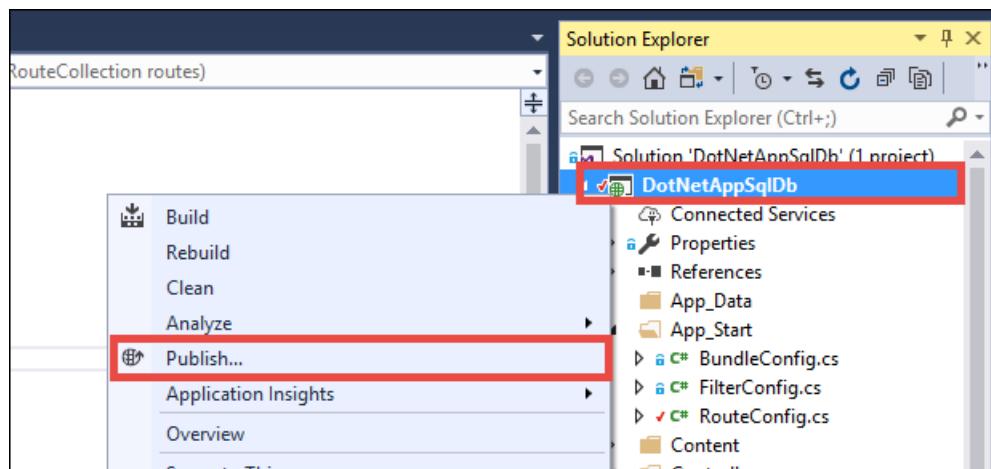
Type **ctrl+F5** to run the app without debugging. The app is displayed in your default browser. Select the **Create New** link and create a couple *to-do* items.

Test the **Edit**, **Details**, and **Delete** links.

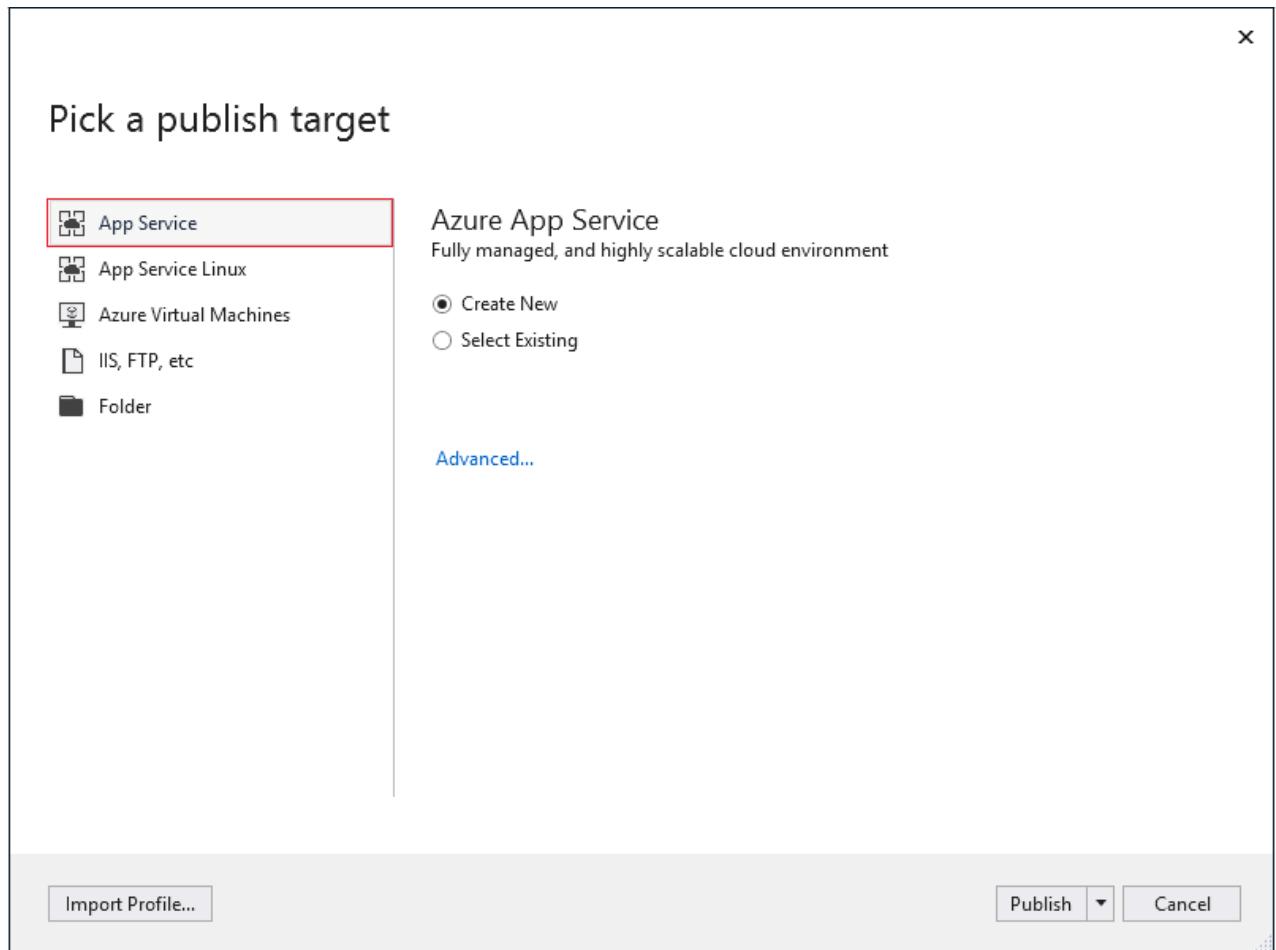
The app uses a database context to connect with the database. In this sample, the database context uses a connection string named **MyDbConnection**. The connection string is set in the *Web.config* file and referenced in the *Models/MyDatabaseContext.cs* file. The connection string name is used later in the tutorial to connect the Azure app to an Azure SQL Database.

Publish to Azure with SQL Database

In the **Solution Explorer**, right-click your **DotNetAppSqlDb** project and select **Publish**.



Make sure that **Microsoft Azure App Service** is selected and click **Publish**.



Publishing opens the **Create App Service** dialog, which helps you create all the Azure resources you need to run your ASP.NET app in Azure.

Sign in to Azure

In the **Create App Service** dialog, click **Add an account**, and then sign in to your Azure subscription. If you're already signed into a Microsoft account, make sure that account holds your Azure subscription. If the signed-in Microsoft account doesn't have your Azure subscription, click it to add the correct account.

NOTE

If you're already signed in, don't select **Create** yet.



Create App Service

Host your web and mobile applications, REST APIs, and more in Azure



You need to be signed in with an Azure account to create a new App Service

[Create your free Azure Account](#)

Already have an account?

[Sign In](#)

[Export...](#)

[Cancel](#)



Configure the web app name

You can keep the generated web app name, or change it to another unique name (valid characters are `a-z`, `0-9`, and `-`). The web app name is used as part of the default URL for your app (`<app_name>.azurewebsites.net`, where `<app_name>` is your web app name). The web app name needs to be unique across all apps in Azure.

Create App Service

Host your web and mobile applications, REST APIs, and more in Azure

App Name
DotNetAppSqlDb12

Subscription
Visual Studio Enterprise

Resource Group
DotNetAppSqlDb20180625094132ResourceGroup* [New...](#)

Hosting Plan
DotNetAppSqlDb20180625094132Plan* (Central US, S1) [New...](#)

Explore additional Azure services

[Create a SQL Database](#)

[Create a storage account](#)

Clicking the Create button will create the following Azure resources

Hosting Plan - DotNetAppSqlDb20180625094132Plan

App Service - DotNetAppSqlDb12

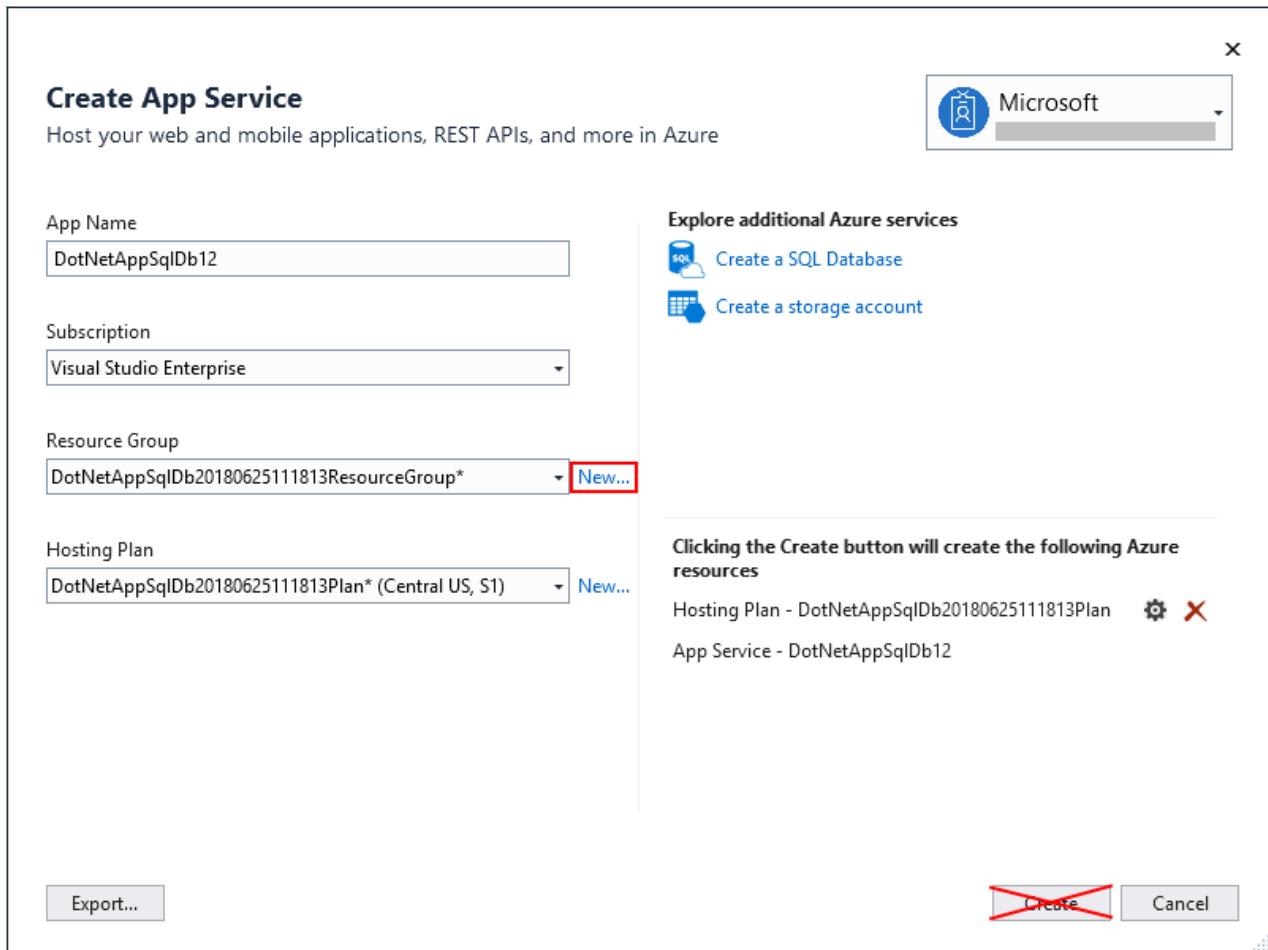
[Export...](#)



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.

Next to **Resource Group**, click **New**.



Name the resource group **myResourceGroup**.

Create an App Service plan

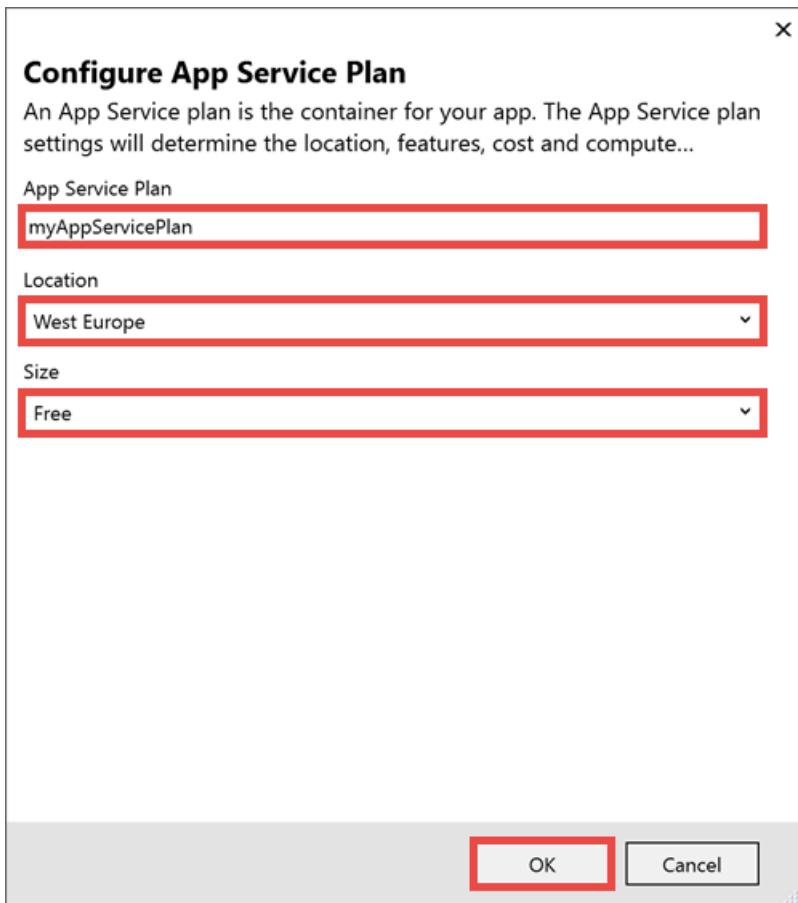
An [App Service plan](#) specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

App Service plans define:

- Region (for example: North Europe, East US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 20 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

Next to **App Service Plan**, click **New**.

In the **Configure App Service Plan** dialog, configure the new App Service plan with the following settings:

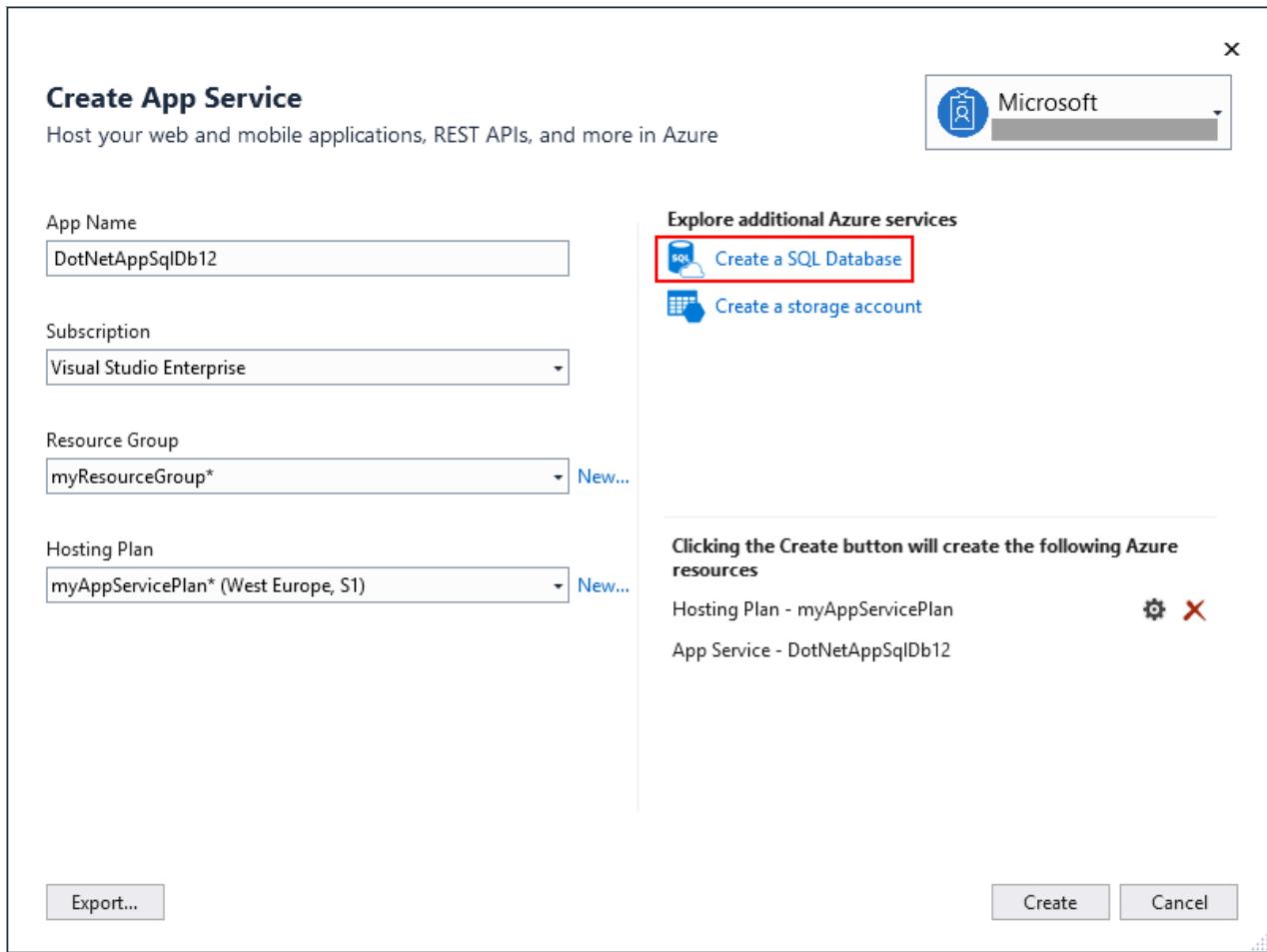


SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
App Service Plan	myAppServicePlan	App Service plans
Location	West Europe	Azure regions
Size	Free	Pricing tiers

Create a SQL Server instance

Before creating a database, you need an [Azure SQL Database logical server](#). A logical server contains a group of databases managed as a group.

Click **Create a SQL Database**.



In the **Configure SQL Database** dialog, click **New** next to **SQL Server**.

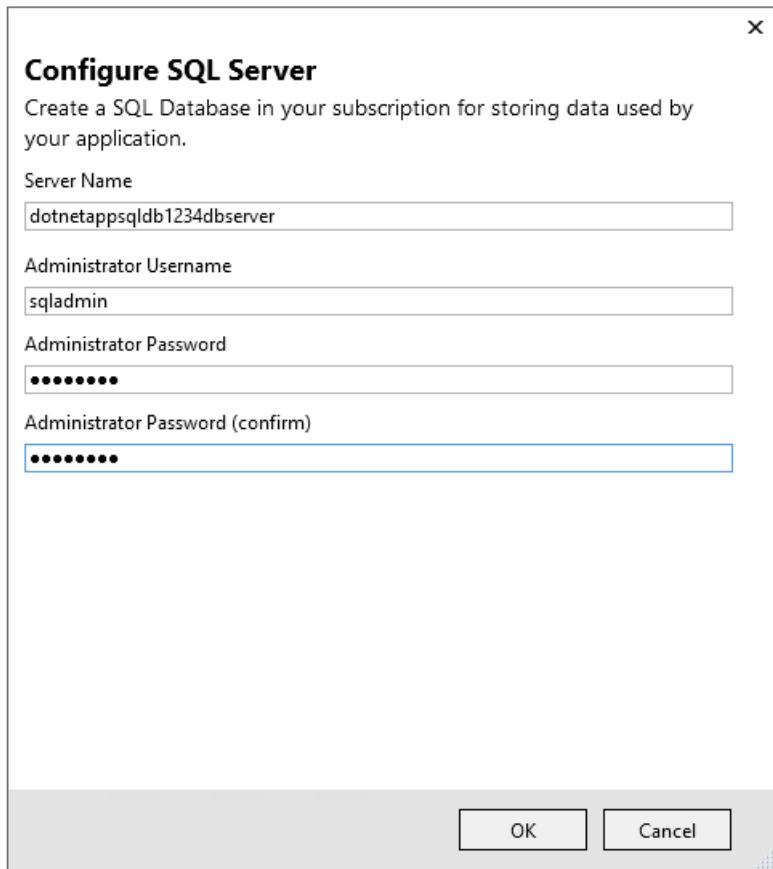
A unique server name is generated. This name is used as part of the default URL for your logical server, `<server_name>.database.windows.net`. It must be unique across all logical server instances in Azure. You can change the server name, but for this tutorial, keep the generated value.

Add an administrator username and password. For password complexity requirements, see [Password Policy](#).

Remember this username and password. You need them to manage the logical server instance later.

IMPORTANT

Even though your password in the connection strings is masked (in Visual Studio and also in App Service), the fact that it's maintained somewhere adds to the attack surface of your app. App Service can use [managed service identities](#) to eliminate this risk by removing the need to maintain secrets in your code or app configuration at all. For more information, see [Next steps](#).

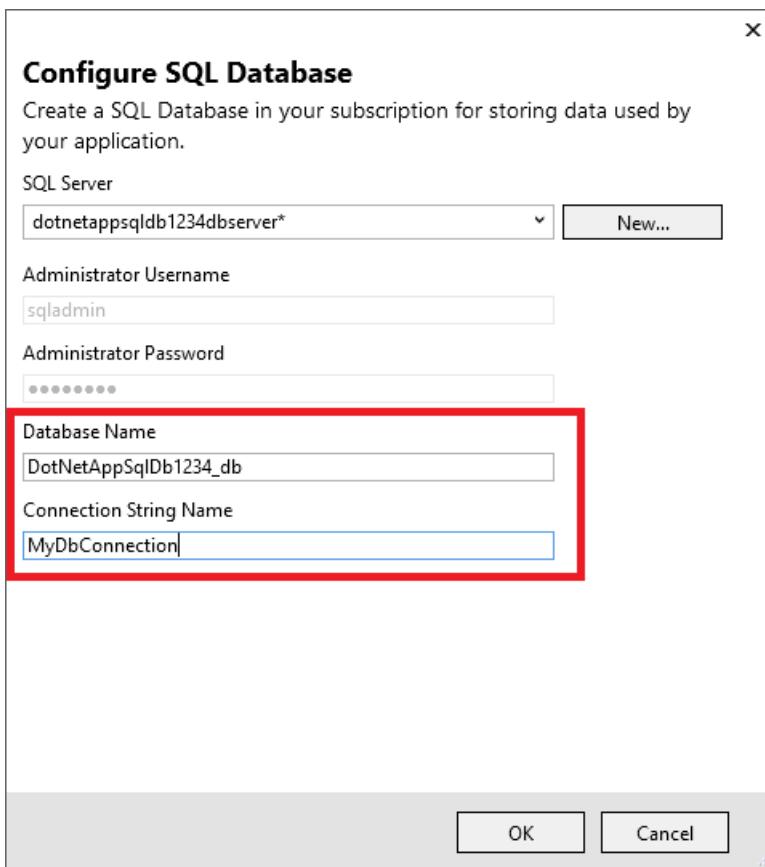


Click **OK**. Don't close the **Configure SQL Database** dialog yet.

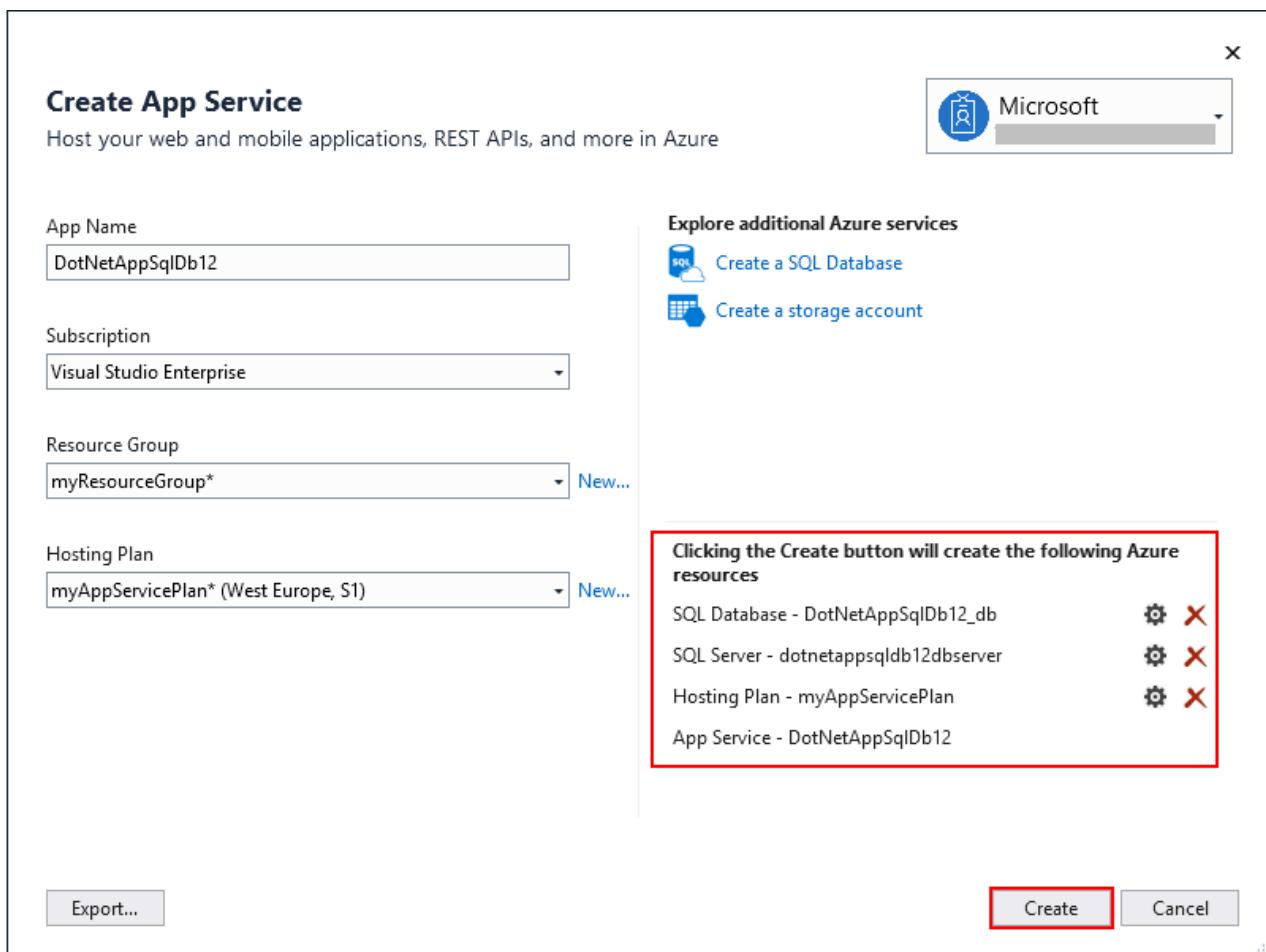
Create a SQL Database

In the **Configure SQL Database** dialog:

- Keep the default generated **Database Name**.
- In **Connection String Name**, type *MyDbConnection*. This name must match the connection string that is referenced in *Models/MyDatabaseContext.cs*.
- Select **OK**.



The **Create App Service** dialog shows the resources you've configured. Click **Create**.



Once the wizard finishes creating the Azure resources, it publishes your ASP.NET app to Azure. Your default browser is launched with the URL to the deployed app.

Add a few to-do items.

Description	Created Date	
Deploy app to Azure	2017-06-01	Edit Details Delete
Walk dog	2017-06-03	Edit Details Delete
Feed cat	2017-06-04	Edit Details Delete

© 2017 - My ASP.NET Application

Congratulations! Your data-driven ASP.NET application is running live in Azure App Service.

Access the SQL Database locally

Visual Studio lets you explore and manage your new SQL Database easily in the **SQL Server Object Explorer**.

Create a database connection

From the **View** menu, select **SQL Server Object Explorer**.

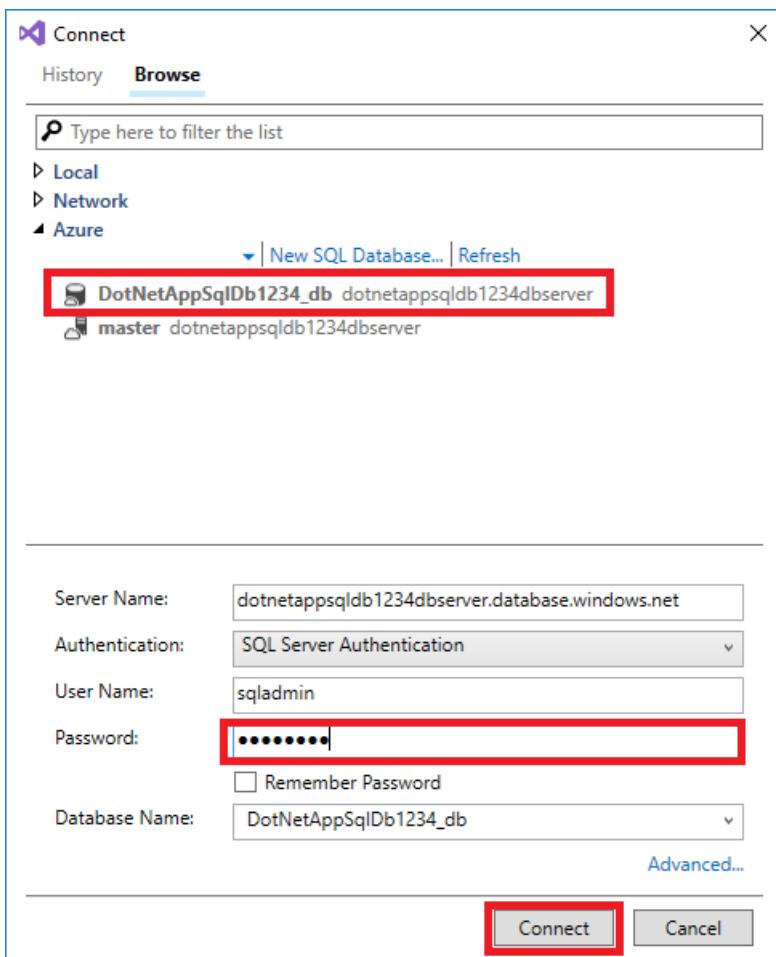
At the top of **SQL Server Object Explorer**, click the **Add SQL Server** button.

Configure the database connection

In the **Connect** dialog, expand the **Azure** node. All your SQL Database instances in Azure are listed here.

Select the SQL Database that you created earlier. The connection you created earlier is automatically filled at the bottom.

Type the database administrator password you created earlier and click **Connect**.

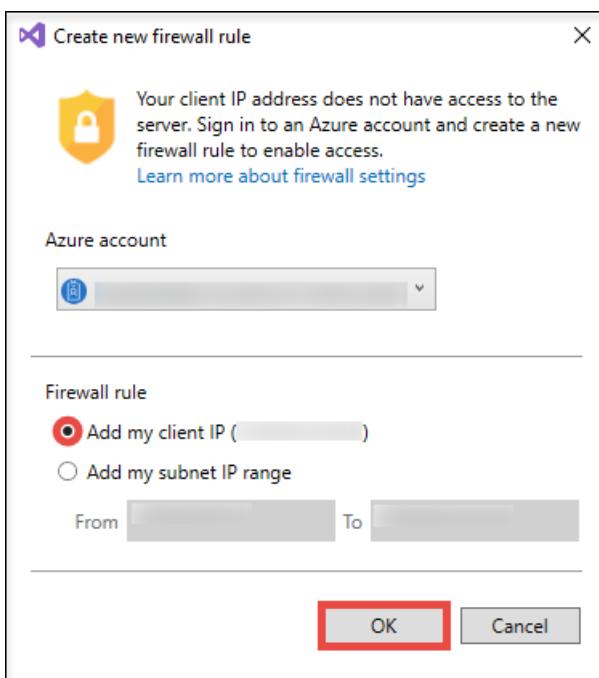


Allow client connection from your computer

The **Create a new firewall rule** dialog is opened. By default, your SQL Database instance only allows connections from Azure services, such as your Azure app. To connect to your database, create a firewall rule in the SQL Database instance. The firewall rule allows the public IP address of your local computer.

The dialog is already filled with your computer's public IP address.

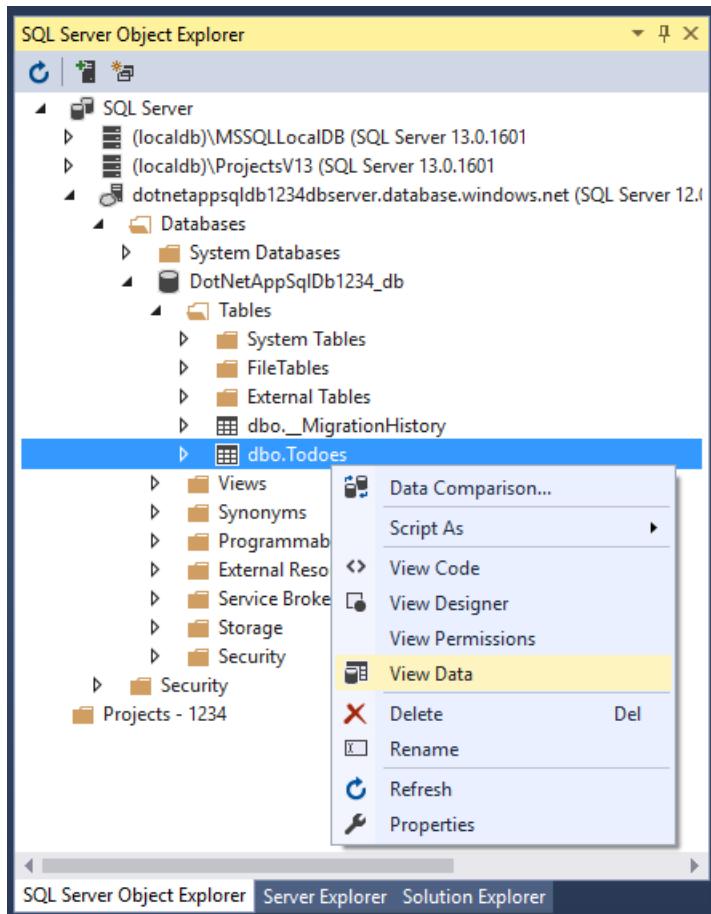
Make sure that **Add my client IP** is selected and click **OK**.



Once Visual Studio finishes creating the firewall setting for your SQL Database instance, your connection shows up in **SQL Server Object Explorer**.

Here, you can perform the most common database operations, such as run queries, create views and stored procedures, and more.

Expand your connection > **Databases** > <your database> > **Tables**. Right-click on the `Todos` table and select **View Data**.



Update app with Code First Migrations

You can use the familiar tools in Visual Studio to update your database and app in Azure. In this step, you use Code First Migrations in Entity Framework to make a change to your database schema and publish it to Azure.

For more information about using Entity Framework Code First Migrations, see [Getting Started with Entity Framework 6 Code First using MVC 5](#).

Update your data model

Open `Models\Todo.cs` in the code editor. Add the following property to the `Todo` class:

```
public bool Done { get; set; }
```

Run Code First Migrations locally

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

From the **Tools** menu, click **NuGet Package Manager** > **Package Manager Console**.

In the Package Manager Console window, enable Code First Migrations:

```
Enable-Migrations
```

Add a migration:

```
Add-Migration AddProperty
```

Update the local database:

```
Update-Database
```

Type `ctrl+F5` to run the app. Test the edit, details, and create links.

If the application loads without errors, then Code First Migrations has succeeded. However, your page still looks the same because your application logic is not using this new property yet.

Use the new property

Make some changes in your code to use the `Done` property. For simplicity in this tutorial, you're only going to change the `Index` and `Create` views to see the property in action.

Open `Controllers\TodosController.cs`.

Find the `Create()` method on line 52 and add `Done` to the list of properties in the `Bind` attribute. When you're done, your `Create()` method signature looks like the following code:

```
public ActionResult Create([Bind(Include = "Description,CreatedDate,Done")] Todo todo)
```

Open `Views\Todos\Create.cshtml`.

In the Razor code, you should see a `<div class="form-group">` element that uses `model.Description`, and then another `<div class="form-group">` element that uses `model.CreatedDate`. Immediately following these two elements, add another `<div class="form-group">` element that uses `model.Done`:

```
<div class="form-group">
    @Html.LabelFor(model => model.Done, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        <div class="checkbox">
            @Html.EditorFor(model => model.Done)
            @Html.ValidationMessageFor(model => model.Done, "", new { @class = "text-danger" })
        </div>
    </div>
</div>
```

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 `Html.ActionLink()` helper methods. Above this `<td>`, add another `<td>` element with the following Razor code:

```
<td>
    @Html.DisplayFor(modelItem => item.Done)
</td>
```

That's all you need to see the changes in the `Index` and `Create` views.

Type `ctrl+F5` to run the app.

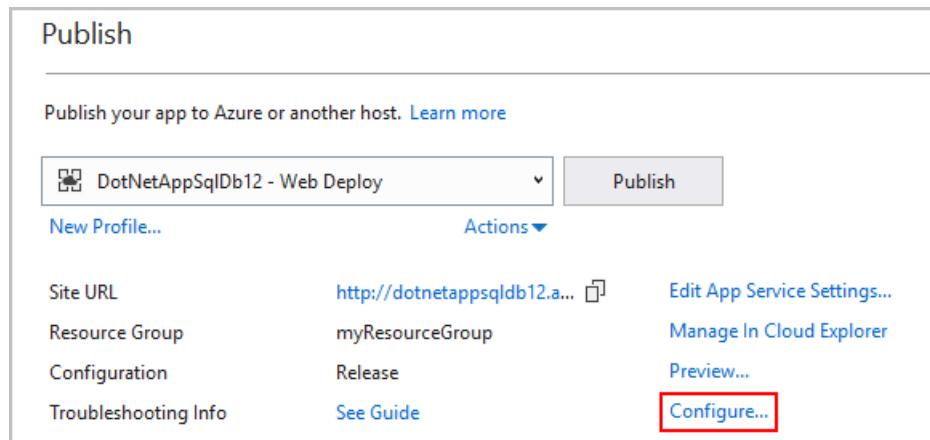
You can now add a to-do item and check **Done**. Then it should show up in your homepage as a completed item. Remember that the `Edit` view doesn't show the `Done` field, because you didn't change the `Edit` view.

Enable Code First Migrations in Azure

Now that your code change works, including database migration, you publish it to your Azure app and update your SQL Database with Code First Migrations too.

Just like before, right-click your project and select **Publish**.

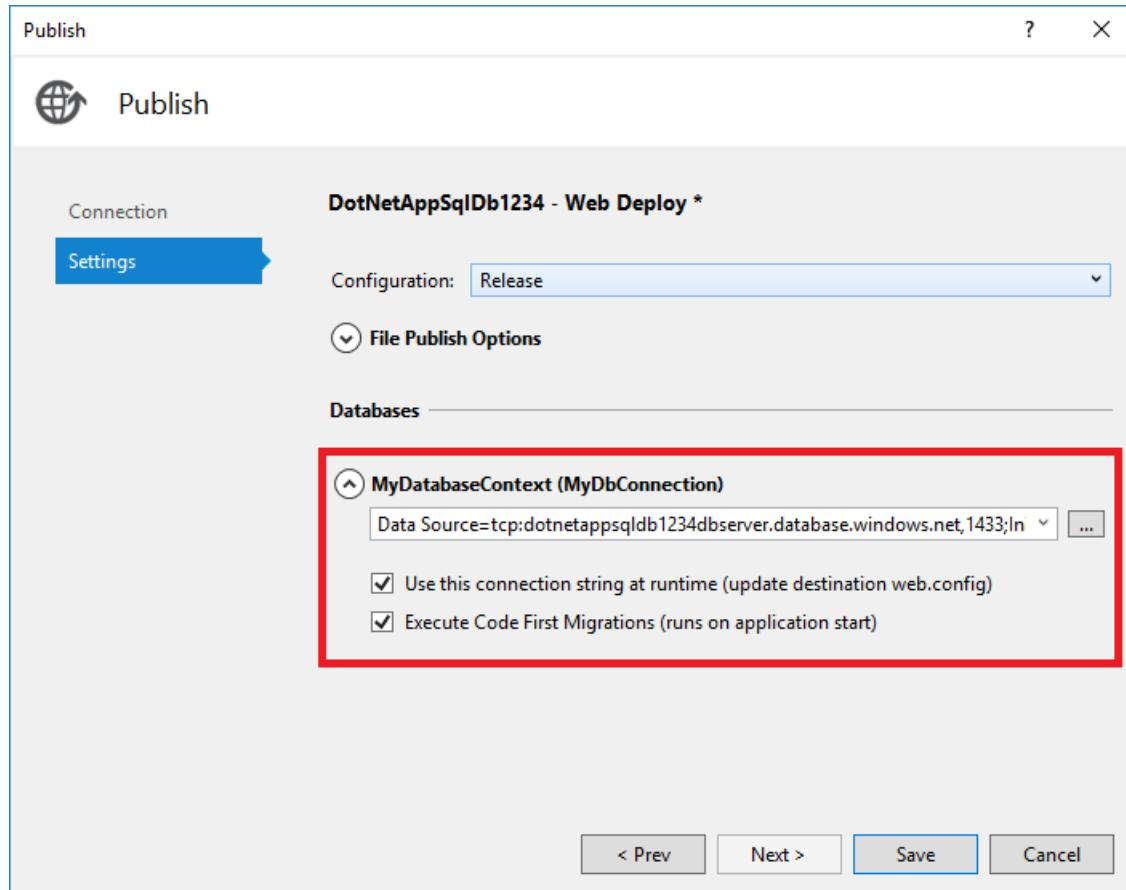
Click **Configure** to open the publish settings.



In the wizard, click **Next**.

Make sure that the connection string for your SQL Database is populated in **MyDbContext** (**MyDbContext**). You may need to select the **myToDoAppDb** database from the dropdown.

Select **Execute Code First Migrations (runs on application start)**, then click **Save**.



Publish your changes

Now that you enabled Code First Migrations in your Azure app, publish your code changes.

In the publish page, click **Publish**.

Try adding to-do items again and select **Done**, and they should show up in your homepage as a completed item.

Description	Created Date	Done	
Deploy app to Azure	2017-06-01	<input checked="" type="checkbox"/>	Edit Details Delete
Walk dog	2017-06-03	<input type="checkbox"/>	Edit Details Delete
Feed cat	2017-06-04	<input type="checkbox"/>	Edit Details Delete

© 2017 - My ASP.NET Application

All your existing to-do items are still displayed. When you republish your ASP.NET application, existing data in your SQL Database is not lost. Also, Code First Migrations only changes the data schema and leaves your existing data intact.

Stream application logs

You can stream tracing messages directly from your Azure app to Visual Studio.

Open `Controllers\TodosController.cs`.

Each action starts with a `Trace.WriteLine()` method. This code is added to show you how to add trace messages to your Azure app.

Open Server Explorer

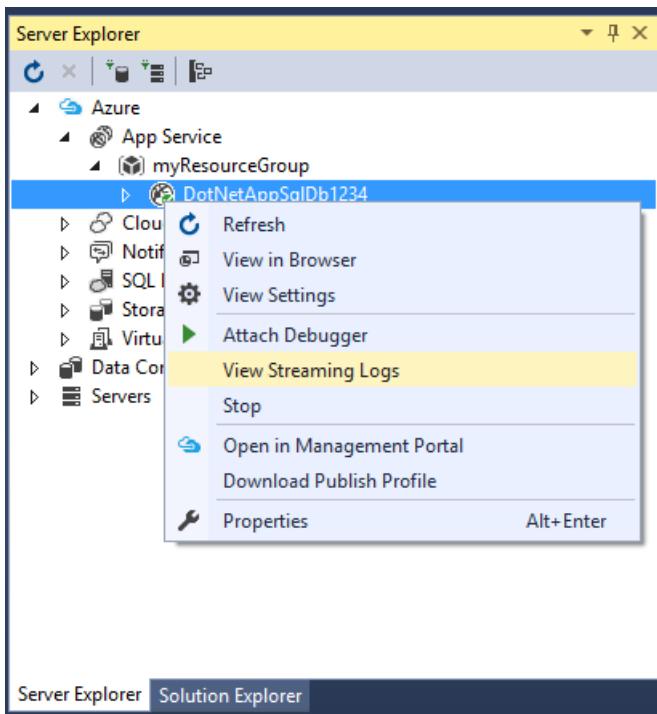
From the **View** menu, select **Server Explorer**. You can configure logging for your Azure app in **Server Explorer**.

Enable log streaming

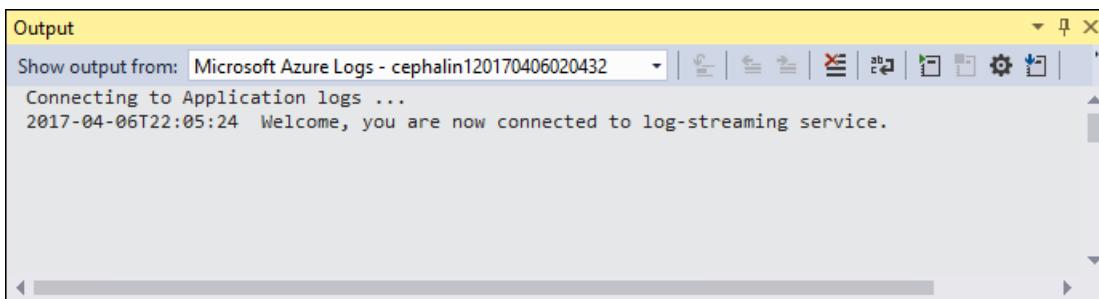
In **Server Explorer**, expand **Azure > App Service**.

Expand the **myResourceGroup** resource group, you created when you first created the Azure app.

Right-click your Azure app and select **View Streaming Logs**.



The logs are now streamed into the **Output** window.



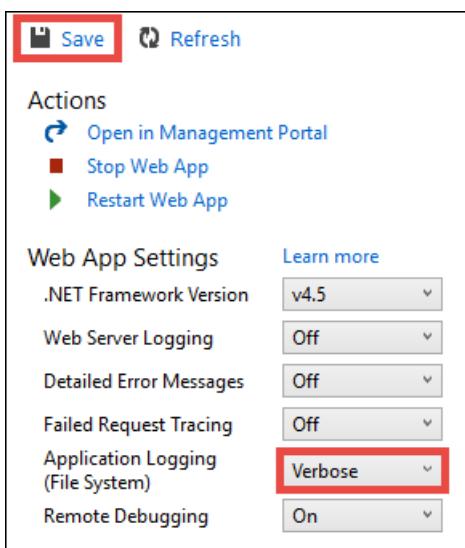
However, you don't see any of the trace messages yet. That's because when you first select **View Streaming Logs**, your Azure app sets the trace level to `Error`, which only logs error events (with the `Trace.TraceError()` method).

Change trace levels

To change the trace levels to output other trace messages, go back to **Server Explorer**.

Right-click your Azure app again and select **View Settings**.

In the **Application Logging (File System)** dropdown, select **Verbose**. Click **Save**.



TIP

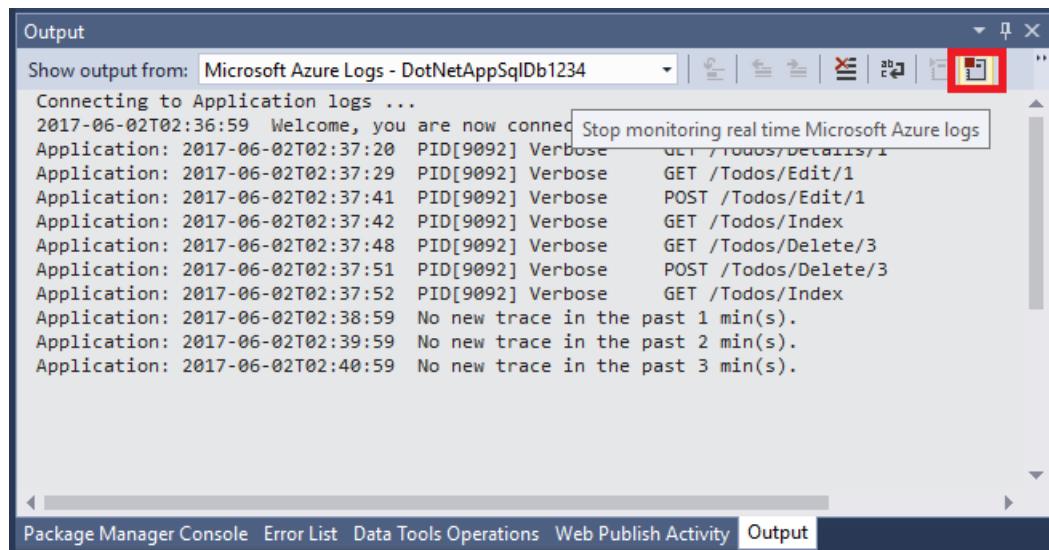
You can experiment with different trace levels to see what types of messages are displayed for each level. For example, the **Information** level includes all logs created by `Trace.TraceInformation()`, `Trace.TraceWarning()`, and `Trace.TraceError()`, but not logs created by `Trace.WriteLine()`.

In your browser navigate to your app again at `http://<your app name>.azurewebsites.net`, then try clicking around the to-do list application in Azure. The trace messages are now streamed to the **Output** window in Visual Studio.

```
Application: 2017-04-06T23:30:41 PID[8132] Verbose      GET /Todos/Index
Application: 2017-04-06T23:30:43 PID[8132] Verbose      GET /Todos/Create
Application: 2017-04-06T23:30:53 PID[8132] Verbose      POST /Todos/Create
Application: 2017-04-06T23:30:54 PID[8132] Verbose      GET /Todos/Index
```

Stop log streaming

To stop the log-streaming service, click the **Stop monitoring** button in the **Output** window.



Manage your Azure app

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

A screenshot of the Azure portal search results. The search bar at the top contains the text "App Services". Below the search bar, a list of services is shown under the heading "Services". The first item in the list is "App Services", which is highlighted with a blue background. Other items include "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 blue icon to its left.

Select the name of your Azure app.

The screenshot shows the Azure portal's 'Subscriptions' view. At the top, there are buttons for 'Add', 'Columns', and 'Refresh'. Below that, a search bar says 'Filter by name...' and dropdown menus for 'All subscriptions', 'All locations', and 'No grouping'. A message indicates '1 items'. A table lists one item: 'NAME' (DotNetAppSqlDb1234), 'RESOURCE GR...' (myResourceGroup), 'STATUS' (Running), 'APP TYPE' (Web app), and 'APP SERVICE PLAN' (myAppServicePlan). The 'DotNetAppSqlDb1234' entry is highlighted with a red box.

You have landed in your app's page.

By default, the portal shows the **Overview** page. This page gives you a view of how your app is doing. Here, you can also perform basic management tasks like browse, stop, start, restart, and delete. The tabs on the left side of the page show the different configuration pages you can open.

The screenshot shows the Azure portal's 'Overview' page for a web app. On the left, a sidebar lists navigation options: Overview (selected and highlighted in blue), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview), Application settings, Authentication / Authorization, Backups, and Custom domains. The main content area shows the following details:

Resource group (change)	URL
myResourceGroup	http://dotnetappsqldb1234.azurewebsites.net
Status	App Service plan/pricing tier
Running	myAppServicePlan (Free)
Location	FTP/deployment username
West Europe	DotNetAppSqlDb1234\rick@!#\$%
Subscription name (change)	FTP hostname
MSDN	ftp://waws-prod-am2-109.ftp.azurewebsites.window...
Subscription ID	FTPS hostname
74812a68-74812a68-74812a68-8422aec1	ftps://waws-prod-am2-109.ftp.azurewebsites.window...

The 'Monitoring' section contains a chart titled 'Requests and errors' showing activity from 4:15 PM to 5 PM. The Y-axis ranges from 0 to 5, and the X-axis shows time points at 4:15 PM, 4:30 PM, 4:45 PM, and 5 PM. Blue bars represent 'REQUESTS' and red bars represent 'HTTP SERVER ERRORS'. There is a single request at 4:15 PM and two requests at 4:45 PM, with no errors.

Clean up resources

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

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

Next steps

In this tutorial, you learned how to:

- Create a SQL Database in Azure
- Connect an ASP.NET app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to easily improve the security of your connection Azure SQL Database.

[Access SQL Database securely using managed identities for Azure resources](#)

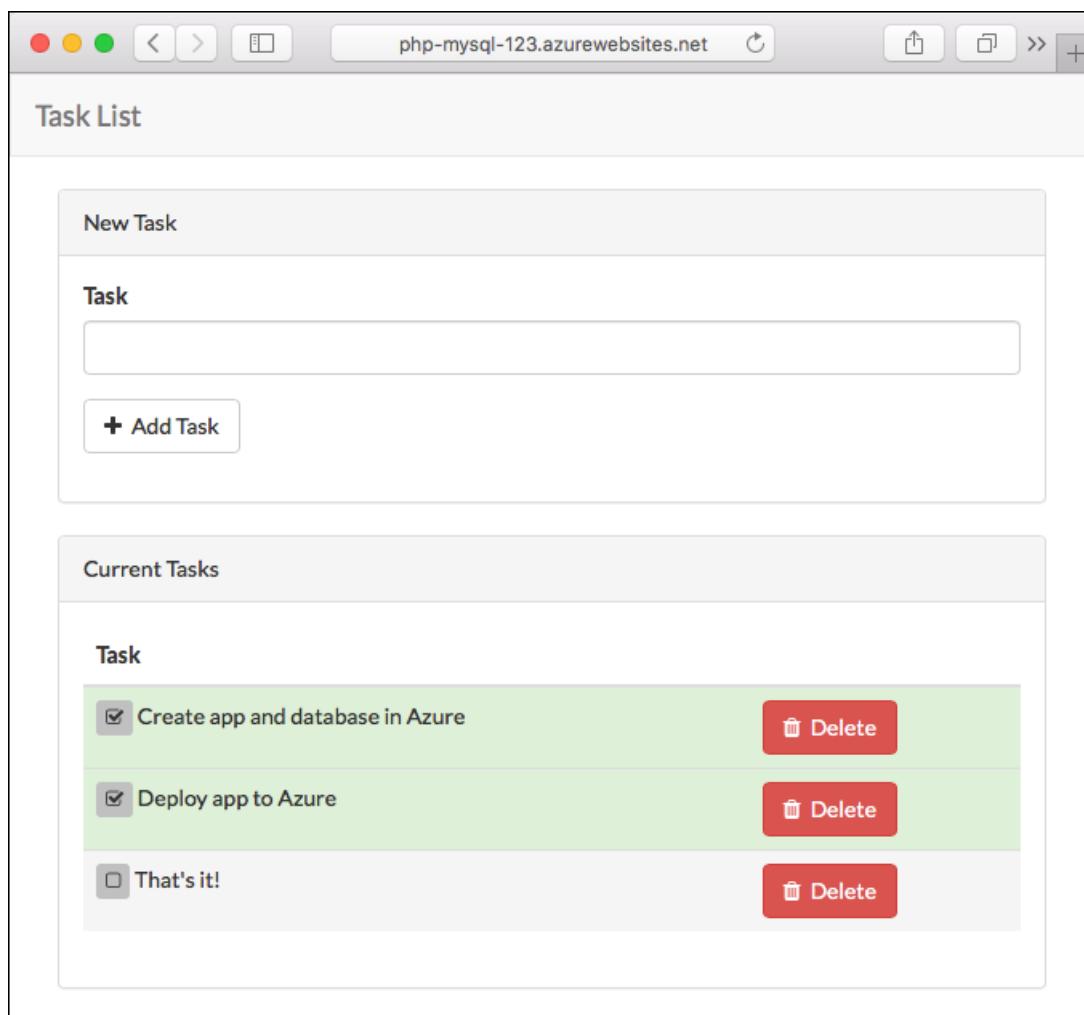
Tutorial: Build a PHP and MySQL app in Azure

2/20/2020 • 17 minutes to read • [Edit Online](#)

NOTE

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

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



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 text file named `.env`. Copy the following variables into the `.env` file. Replace the `<root_password>` placeholder with the MySQL root user's password.

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_DATABASE=sampledbs
DB_USERNAME=root
DB_PASSWORD=<root_password>
```

For information on how Laravel uses the `.env` file, see [Laravel Environment Configuration](#).

Run the sample locally

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 titled "Task List" running on a local server at "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
- Deploy data-driven app
- That's it!

Each task entry has a red "Delete" button to its right.

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

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

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

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

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

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 in **Free** tier, run the `az appservice list-locations --sku FREE` command.

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

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

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

Create a MySQL server

In the Cloud Shell, create a server in Azure Database for MySQL with the `az mysql server create` command.

In the following command, substitute a unique server name for the `<mysql_server_name>` placeholder, a user name for the `<admin_user>`, and a password for the `<admin_password>` placeholder. The server name is used as part of your MySQL endpoint (https://<mysql_server_name>.mysql.database.azure.com), so the name needs to be unique across all servers in Azure.

```
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
```

NOTE

Since there are several credentials to think about in this tutorial, to avoid confusion, `--admin-user` and `--admin-password` are set to dummy values. In a production environment, follow security best practices when choosing a good username and password for your MySQL server in Azure.

When the MySQL server is created, the Azure CLI shows information similar to the following example:

```
{  
  "location": "westeurope",  
  "name": "<mysql_server_name>",  
  "resourceGroup": "myResourceGroup",  
  "sku": {  
    "additionalProperties": {},  
    "capacity": 1,  
    "family": "Gen5",  
    "name": "B_Gen5_1",  
    "size": null,  
    "tier": "GeneralPurpose"  
  },  
  "sslEnforcement": "Enabled",  
  ... +  
- < Output has been truncated for readability >  
}
```

Configure server firewall

In the Cloud Shell, create a firewall rule for your MySQL server to allow client connections by using the `az mysql server firewall-rule create` command. When both starting IP and end IP are set to 0.0.0.0, the firewall is only opened for other Azure resources.

```
az mysql server firewall-rule create --name allAzureIPs --server <mysql_server_name> --resource-group myResourceGroup --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0
```

TIP

You can be even more restrictive in your firewall rule by [using only the outbound IP addresses your app uses](#).

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 local terminal window, connect to the MySQL server in Azure. Use the value you specified previously for `<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. Again, for simplicity of the tutorial, use `MySQLAzure2017` as the password.

```
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>` in both `DB_HOST` and `DB_USERNAME`.

```
APP_ENV=production
APP_DEBUG=true
APP_KEY=

DB_CONNECTION=mysql
DB_HOST=<mysql_server_name>.mysql.database.azure.com
DB_DATABASE=sampledb
DB_USERNAME=phpappuser@<mysql_server_name>
DB_PASSWORD=MySQLAzure2017
MYSQL_SSL=true
```

Save the changes.

TIP

To secure your MySQL connection information, this file is already excluded from the Git repository (See `.gitignore` in the repository root). Later, you learn how to configure environment variables in App Service to connect to your database in Azure Database for MySQL. With environment variables, you don't need the `.env` file in App Service.

Configure 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')) ? [
        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.

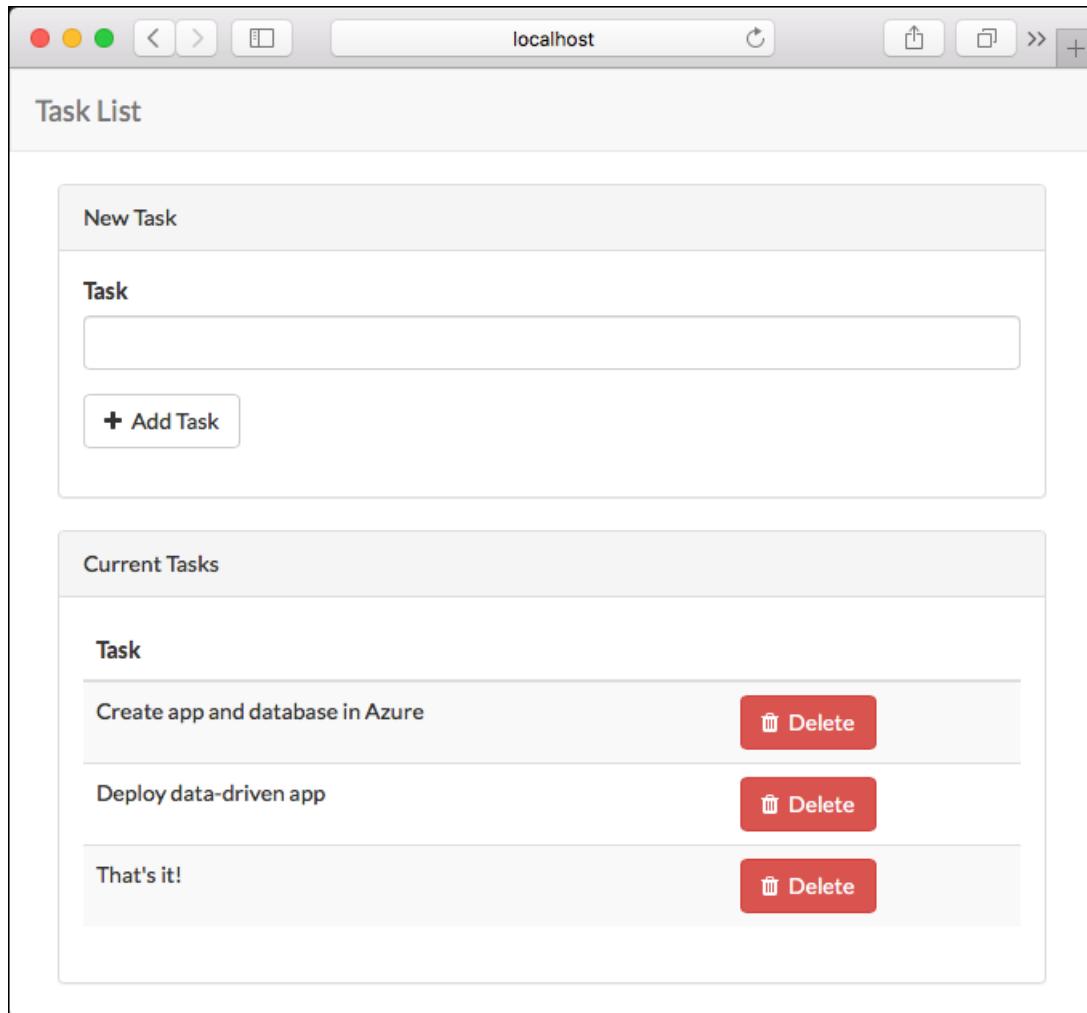
```
php artisan key:generate --env=production --force
```

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.



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

Commit your changes

Run the following Git commands to commit your changes:

```
git add .
git commit -m "database.php updates"
```

Your app is ready to be deployed.

Deploy to Azure

In this step, you deploy the MySQL-connected PHP application to Azure App Service.

Configure a deployment user

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

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

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

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

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

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

Create an App Service plan

In the Cloud Shell, create an App Service plan with the [az appservice plan create](#) command.

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

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

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

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

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](#).

```
# 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.

Configure database settings

As pointed out previously, you can connect to your Azure MySQL database using environment variables in App Service.

In the Cloud Shell, 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 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.

In the local terminal window, use `php artisan` to generate a new application key without saving it to `.env`.

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

In the Cloud Shell, set the application key in the App Service app by using the `az webapp config appsettings set` command. Replace the placeholders `<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.

Set the virtual application path

Set the virtual application path for the app. This step is required because the [Laravel application lifecycle](#) begins in the `public` directory instead of the application's root directory. Other PHP frameworks whose lifecycle start in the root directory can work without manual configuration of the virtual application path.

In the Cloud Shell, set the virtual application path by using the `az resource update` command. Replace the `<appname>` placeholder.

```
az resource update --name web --resource-group myResourceGroup --namespace Microsoft.Web --resource-type config --parent sites/<app_name> --set properties.virtualApplications[0].physicalPath="site\wwwroot\public" --api-version 2015-06-01
```

By default, Azure App Service points the root virtual application path (`/`) to the root directory of the deployed application files (`sites\wwwroot`).

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: 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 >
```

NOTE

You may notice that the deployment process installs [Composer](#) packages at the end. App Service does not run these automations during default deployment, so this sample repository has three additional files in its root directory to enable it:

- `.deployment` - This file tells App Service to run `bash deploy.sh` as the custom deployment script.
- `deploy.sh` - The custom deployment script. If you review the file, you will see that it runs `php composer.phar install` after `npm install`.
- `composer.phar` - The Composer package manager.

You can use this approach to add any step to your Git-based deployment to App Service. For more information, see [Custom Deployment Script](#).

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 browser window with the URL `php-mysql-123.azurewebsites.net` in the address bar. The page displays a task management interface. At the top, there's a header bar with standard browser controls (back, forward, search, etc.). Below the header is a section titled "Task List" containing a "New Task" form with a text input field and a "Add Task" button. Below this is a "Current Tasks" section listing 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.

Task	Action
Create app and database in Azure	Delete
Deploy app to Azure	Delete
That's it!	Delete

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 local terminal window, 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 local terminal window, run Laravel database migrations to make the change in the local database.

```
php artisan migrate
```

Based on the [Laravel naming convention](#), the model `Task` (see `app/Task.php`) maps to the `tasks` table by default.

Update application logic

Open the `routes/web.php` file. The application defines its routes and business logic here.

At the end of the file, add a route with the following code:

```
/**  
 * Toggle Task completeness  
 */  
Route::post('/task/{id}', function ($id) {  
    error_log('INFO: post /task/'.$id);  
    $task = Task::findOrFail($id);  
  
    $task->complete = !$task->complete;  
    $task->save();  
  
    return redirect('/');  
});
```

The preceding code makes a simple update to the data model by toggling the value of `complete`.

Update the view

Open the `resources/views/tasks.blade.php` file. Search for 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>
```

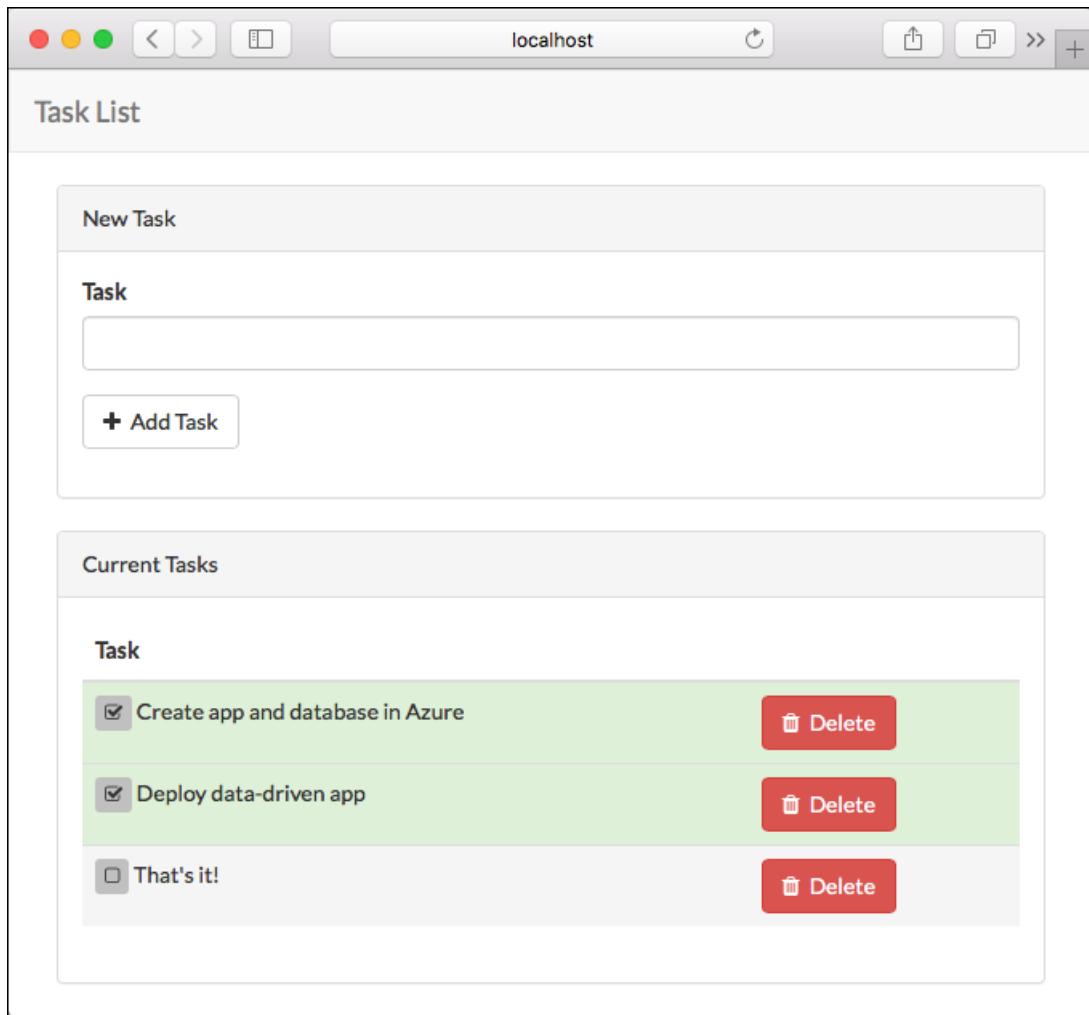
The preceding code adds the submit button that references the route that you defined earlier.

Test the changes locally

In the local terminal window, run the development server from the root directory of the Git repository.

```
php artisan serve
```

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



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

Publish changes to Azure

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

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

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 interface for managing tasks. At the top, the browser title is "php-mysql-123.azurewebsites.net". The main content area is titled "Task List". It contains two sections: "New Task" and "Current Tasks". The "New Task" section has a text input field and a button labeled "+ Add Task". The "Current Tasks" section lists three items, each with a delete button:

- Create app and database in Azure Delete
- Deploy app to Azure Delete
- That's it! Delete

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

Stream diagnostic logs

While the PHP application runs in Azure App Service, you can get the console logs piped to your terminal. That way, you can get the same diagnostic messages to help you debug application errors.

To start log streaming, use the `az webapp log tail` command in the Cloud Shell.

```
az webapp log tail --name <app_name> --resource-group myResourceGroup
```

Once log streaming has started, refresh the Azure app in the browser to get some web traffic. You can now see console logs piped to the terminal. If you don't see console logs immediately, check again in 30 seconds.

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

TIP

A PHP application can use the standard `error_log()` to output to the console. The sample application uses this approach in `app/Http/routes.php`.

As a web framework, [Laravel uses Monolog](#) as the logging provider. To see how to get Monolog to output messages to the console, see [PHP: How to use monolog to log to console \(php://out\)](#).

Manage the Azure app

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.

The screenshot shows the Azure portal's 'App Services' blade. On the left, there's a sidebar with icons for 'Web', 'Function', 'API', 'Mobile', 'Data', 'Machine learning', and 'Container'. The 'Web' icon is highlighted with a red box. The main area is titled 'Subscriptions: All 2 selected' and shows a table with one item: 'php-mysql-123'. The table columns are NAME, STATUS, APP TYPE, APP SERVICE PLAN, LOCATION, and SUBSCRIPTION. The details for 'php-mysql-123' are: Status: Running, App Type: Web app, App Service Plan: myAppServicePlan, Location: West Europe, Subscription: Visual Studio

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

The left menu provides pages for configuring your app.

The screenshot shows the Azure portal's 'Overview' page for the 'php-mysql-123' app service. The left sidebar has sections for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Deployment', 'Quickstart', 'Deployment credentials', 'Deployment slots', and 'Deployment options'. The 'Overview' section is selected and highlighted with a blue box. The main pane shows the app's configuration: Resource group: myResourceGroup, URL: http://php-mysql-123.azurewebsites.net, Status: Running, Location: West Europe, Subscription name: Visual Studio Ultimate with MSDN, and Subscription ID. Below this is the 'Monitoring' section, which includes a 'Requests and errors' chart showing 4 requests and 3 errors.

Clean up resources

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

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

This command may take a minute to run.

Next steps

In this tutorial, you learned how to:

- Create a MySQL database in Azure
- Connect a PHP app to MySQL
- Deploy the app to Azure
- Update the data model and redeploy the app

- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to map a custom DNS name to the app.

[Map an existing custom DNS name to Azure App Service](#)

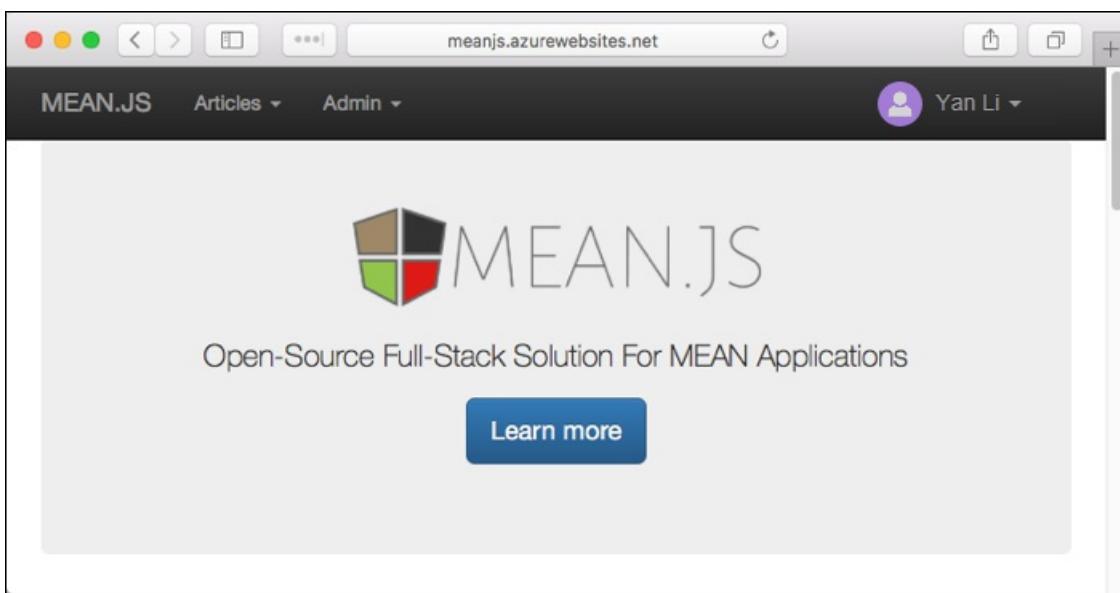
Tutorial: Build a Node.js and MongoDB app in Azure

2/20/2020 • 15 minutes to read • [Edit Online](#)

NOTE

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

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows how to create a Node.js app in App Service and connect it to a MongoDB database. When you're done, you'll have a MEAN application (MongoDB, Express, AngularJS, and Node.js) running in [Azure App Service](#). For simplicity, the sample application uses the [MEAN.js web framework](#).



What you'll learn:

- Create a MongoDB database in Azure
- Connect a Node.js app to MongoDB
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

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

Prerequisites

To complete this tutorial:

1. [Install Git](#)
2. [Install Node.js and NPM](#)
3. [Install Bower](#) (required by [MEAN.js](#))
4. [Install Gulp.js](#) (required by [MEAN.js](#))
5. [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
```

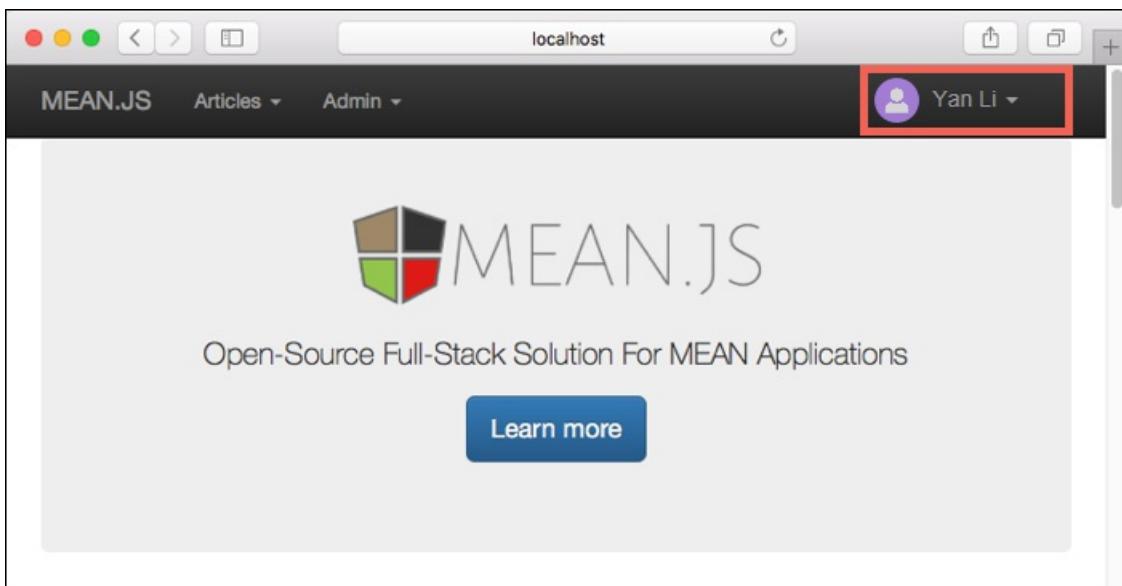
When the app is fully loaded, you see something similar to the following message:

```
-- 
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.

NOTE

The [Node.js quickstart](#) mentions the need for a web.config in the root app directory. However, in this tutorial, this web.config file will be automatically generated by App Service when you deploy your files using [local Git deployment](#) instead of ZIP file deployment.

Use Azure Cloud Shell

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

To start Azure Cloud Shell:

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

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

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

Create production MongoDB

In this step, you create a MongoDB database in Azure. When your app is deployed to Azure, it uses this cloud database.

For MongoDB, this tutorial uses [Azure Cosmos DB](#). Cosmos DB supports MongoDB client connections.

Create a resource group

A [resource group](#) is a logical container into which Azure resources 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 in **Free** tier, run the `az appservice list-locations --sku FREE` command.

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

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

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

Create a Cosmos DB account

NOTE

There is a cost to creating the Azure Cosmos DB databases in this tutorial in your own Azure subscription. To use a free Azure Cosmos DB account for seven days, you can use the [Try Azure Cosmos DB for free](#) experience. Just click the **Create** button in the MongoDB tile to create a free MongoDB database on Azure. Once the database is created, navigate to **Connection String** in the portal and retrieve your Azure Cosmos DB connection string for use later in the tutorial.

In the Cloud Shell, create a Cosmos DB account with the `az cosmosdb create` command.

In the following command, substitute a unique Cosmos DB name for the `<cosmosdb_name>` placeholder. This name is used as the part of the Cosmos DB endpoint, `https://<cosmosdb_name>.documents.azure.com/`, so the name needs to be unique across all Cosmos DB accounts in Azure. The name must contain only lowercase letters, numbers, and the hyphen (-) character, and must be between 3 and 50 characters long.

```
az cosmosdb create --name <cosmosdb_name> --resource-group myResourceGroup --kind MongoDB
```

The `--kind MongoDB` parameter enables MongoDB client connections.

When the Cosmos DB account is created, the Azure CLI shows information similar to the following example:

```
{
  "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 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": "HvtsjIYZ8TwRmIuPEUALRwgKOzJUjW22wPL2U8zoMvhGvregBkBk9LdMTxqBgDETSq7obbwZtdeFY7hElTg==",
  "secondaryMasterKey": "Lu9aeZTiXU4PjuuyGBvgS1N9IRG3oegIrIh95U6Vostf9bJiiIpw3IfwSUgQWSEYM3VeEyrhHJ4rn3Ci0vuFqA==",
  "secondaryReadOnlyMasterKey": "LpsCicpVZqHRy7qbMgrzbRKjbYCwCKPQR10QpgReAOxMcggTvxJFA94fTi0oQ7xtxpftTJcXkjTirQ0pT7QFrQ=="
}
```

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`. By default, `.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

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
```

Run the following command to use the connection string you configured in `config/env/local-production.js`.

```
# Bash  
NODE_ENV=production node server.js  
  
# Windows PowerShell  
$env: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 MongoDB-connected Node.js application to Azure App Service.

Configure a deployment user

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

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

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

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

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

If you get a `'Bad Request'. Details: 400` error, use a stronger password.

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

Create an App Service plan

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

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

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

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

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

Create a web app

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

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `NODE|6.9`. To see all supported runtimes, run `az webapp list-runtimes`.

```
# 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 '6c7c716eee'.  
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.

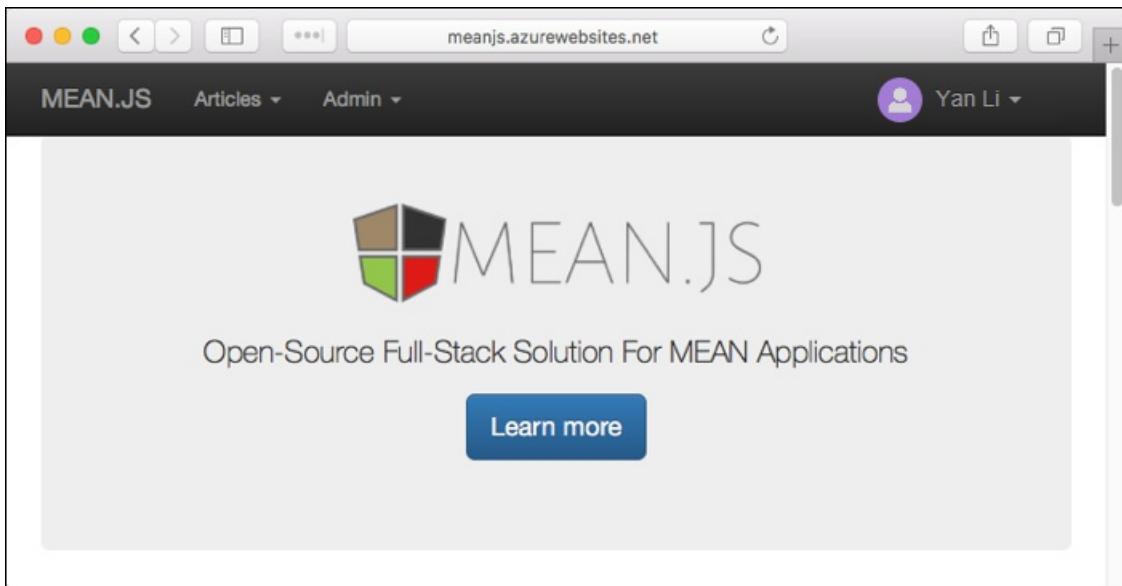
Browse to the Azure app

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 MongoDB (Cosmos DB) database.



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

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

Update data model and redeploy

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

Update the data model

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:

```
const 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 `` 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 `` 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.

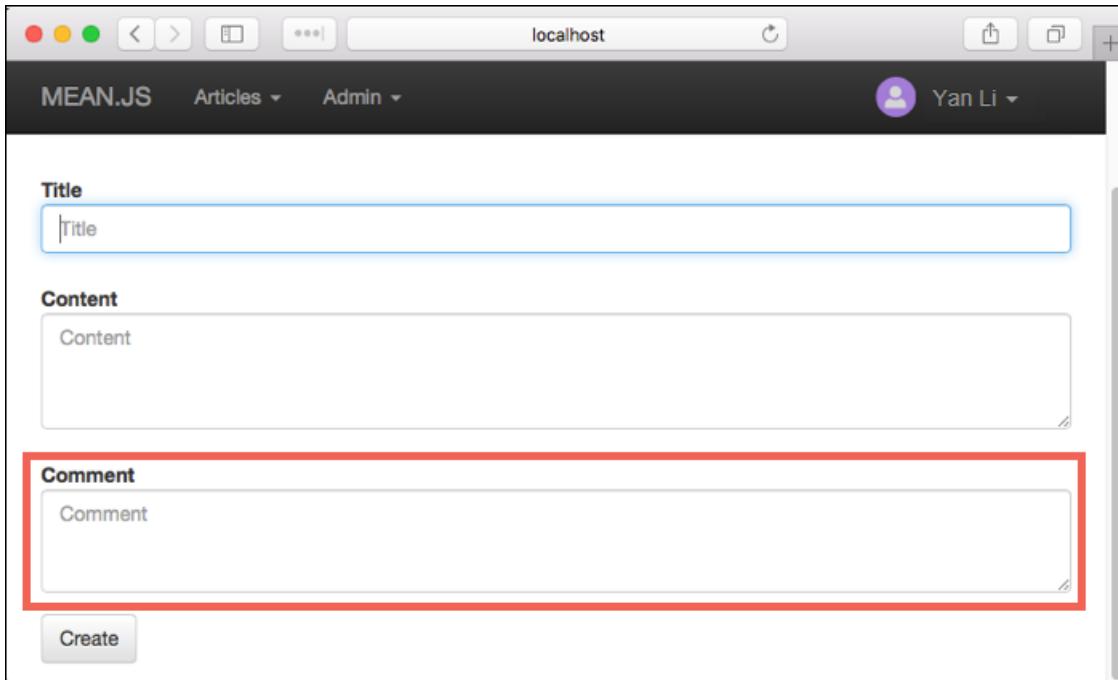
```
# Bash
gulp prod
NODE_ENV=production node server.js

# Windows PowerShell
gulp prod
$env: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

In the local terminal window, 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 application interface for creating a new article. At the top, there's a navigation bar with links for 'MEAN.JS', 'Articles', 'Admin', and a user profile for 'Yan Li'. Below the navigation, the main content area has a heading 'New Article'. It contains three input fields: 'Title' (with placeholder 'Title'), 'Content' (with placeholder 'Content'), and 'Comment' (with placeholder 'Comment'). The 'Comment' field is highlighted with a red border. At the bottom of the form 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

While your Node.js application runs in Azure App Service, you can get the console logs piped to your terminal. That way, you can get the same diagnostic messages to help you debug application errors.

To start log streaming, use the `az webapp log tail` command in the Cloud Shell.

```
az webapp log tail --name <app_name> --resource-group myResourceGroup
```

Once log streaming has started, refresh your Azure app in the browser to get some web traffic. You now see console logs piped to your terminal.

Stop log streaming at any time by typing `ctrl+c`.

Manage your Azure app

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

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

The screenshot shows the Azure App Services portal. On the left, there's a sidebar with icons for Add, Columns, Refresh, and other services like Storage, Functions, and Logic Apps. The main area displays a table of app services. The first row, 'meanjs', is highlighted with a red box. The columns are labeled NAME, STATUS, APP TYPE, APP SERVICE PLAN, and LOCATION. The details for 'meanjs' are: NAME: meanjs, STATUS: Running, APP TYPE: Web app, APP SERVICE PLAN: myAppServicePlan, LOCATION: West Europe.

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 includes tabs for Overview (which is selected and highlighted in blue), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart and Deployment credentials), and a search bar. The main content area has tabs for Browse, Stop, Swap, Restart, Delete, and More. Below these are sections for Essentials and Monitoring. The Essentials section contains details like Resource group (myResourceGroup), Status (Running), Location (West Europe), Subscription name (myAppServicePlan), and URLs (http://meanjs.azurewebsites.net and ftp://waws-prod-am2-025.ftp.azurewebsites.net). The Monitoring section shows Requests and errors.

Clean up resources

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

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

This command may take a minute to run.

Next steps

What you learned:

- Create a MongoDB database in Azure
- Connect a Node.js app to MongoDB
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

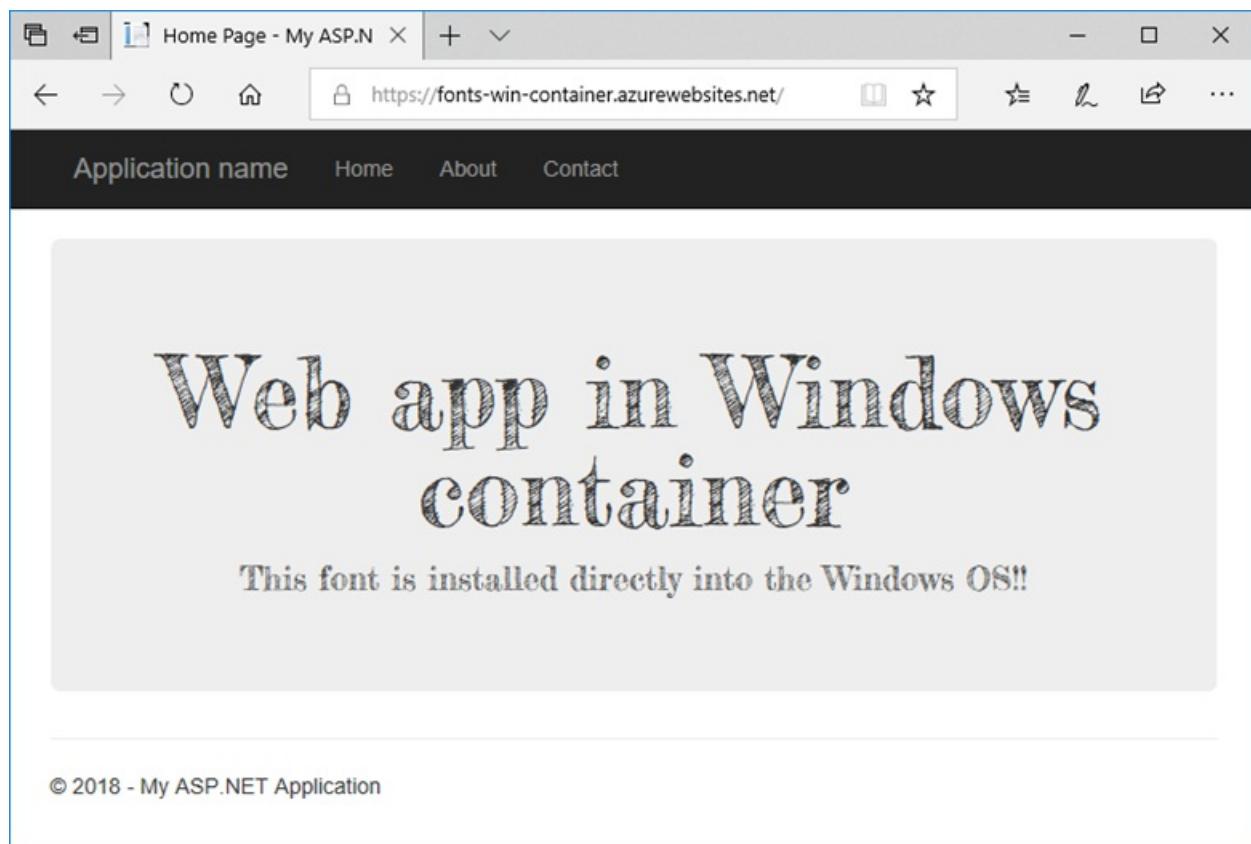
Advance to the next tutorial to learn how to map a custom DNS name to the app.

[Map an existing custom DNS name to Azure App Service](#)

Migrate an ASP.NET app to Azure App Service using a Windows container (Preview)

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

Azure App Service provides pre-defined application stacks on Windows like ASP.NET or Node.js, running on IIS. The preconfigured Windows environment locks down the operating system from administrative access, software installations, changes to the global assembly cache, and so on (see [Operating system functionality on Azure App Service](#)). However, using a custom Windows container in App Service lets you make OS changes that your app needs, so it's easy to migrate on-premises app that requires custom OS and software configuration. This tutorial demonstrates how to migrate to App Service an ASP.NET app that uses custom fonts installed in the Windows font library. You deploy a custom-configured Windows image from Visual Studio to [Azure Container Registry](#), and then run it in App Service.



Prerequisites

To complete this tutorial:

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

Set up the app locally

Download the sample

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

- [Download the sample project](#).
- Extract (unzip) the *custom-font-win-container.zip* file.

The sample project contains a simple ASP.NET application that uses a custom font that is installed into the Windows font library. It's not necessary to install fonts, but it's an example of an app that is integrated with the underlying OS. To migrate such an app to App Service, you either rearchitect your code to remove the integration, or migrate it as-is in a custom Windows container.

Install the font

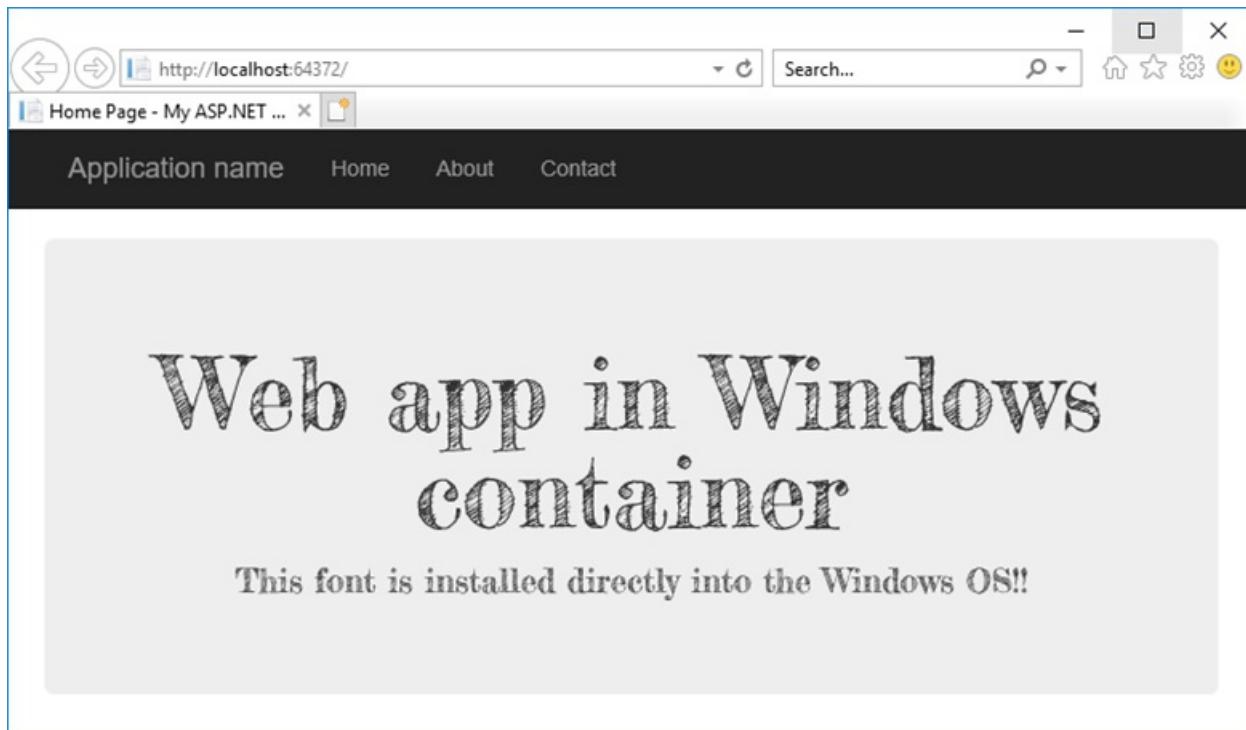
In Windows Explorer, navigate to *custom-font-win-container-master/CustomFontSample*, right-click *FredericktheGreat-Regular.ttf*, and select **Install**.

This font is publicly available from [Google Fonts](#).

Run the app

Open the *custom-font-win-container/CustomFontSample.sln* file in Visual Studio.

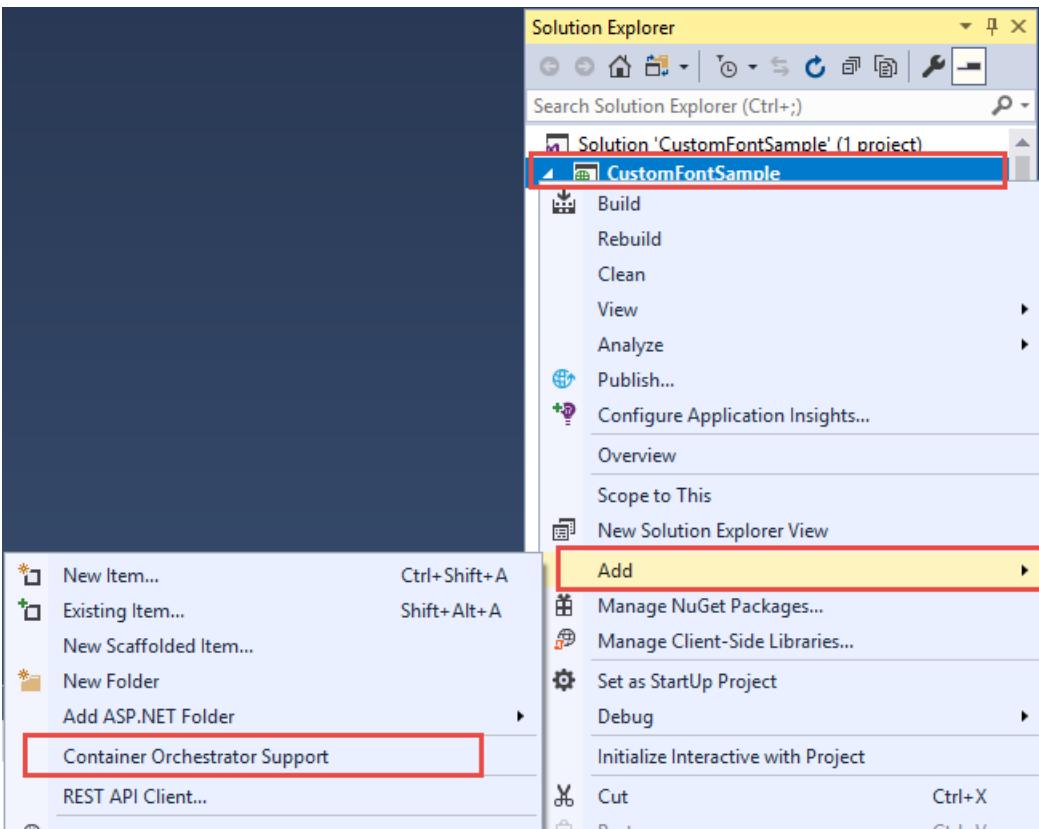
Type `ctrl+F5` to run the app without debugging. The app is displayed in your default browser.



Because it uses an installed font, the app can't run in the App Service sandbox. However, you can deploy it using a Windows container instead, because you can install the font in the Windows container.

Configure Windows container

In Solution Explorer, right-click the **CustomFontSample** project and select **Add > Container Orchestration Support**.



Select **Docker Compose** > **OK**.

Your project is now set up to run in a Windows container. A *Dockerfile* is added to the **CustomFontSample** project, and a **docker-compose** project is added to the solution.

From the Solution Explorer, open **Dockerfile**.

You need to use a [supported parent image](#). Change the parent image by replacing the `FROM` line with the following code:

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

At the end of the file, add the following line and save the file:

```
RUN ${source:-obj/Docker/publish/InstallFont.ps1}
```

You can find *InstallFont.ps1* in the **CustomFontSample** project. It's a simple script that installs the font. You can find a more complex version of the script in the [Script Center](#).

NOTE

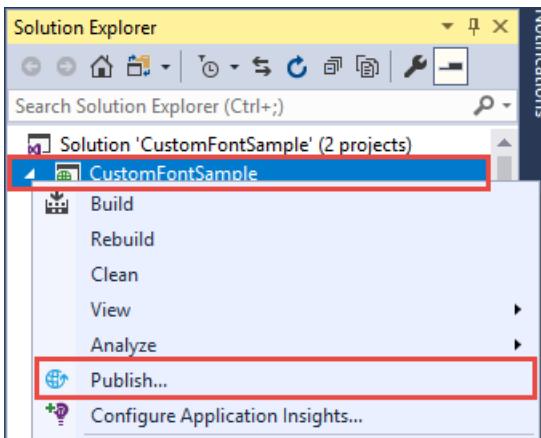
To test the Windows container locally, make sure that Docker is started on your local machine.

Publish to Azure Container Registry

[Azure Container Registry](#) can store your images for container deployments. You can configure App Service to use images hosted in Azure Container Registry.

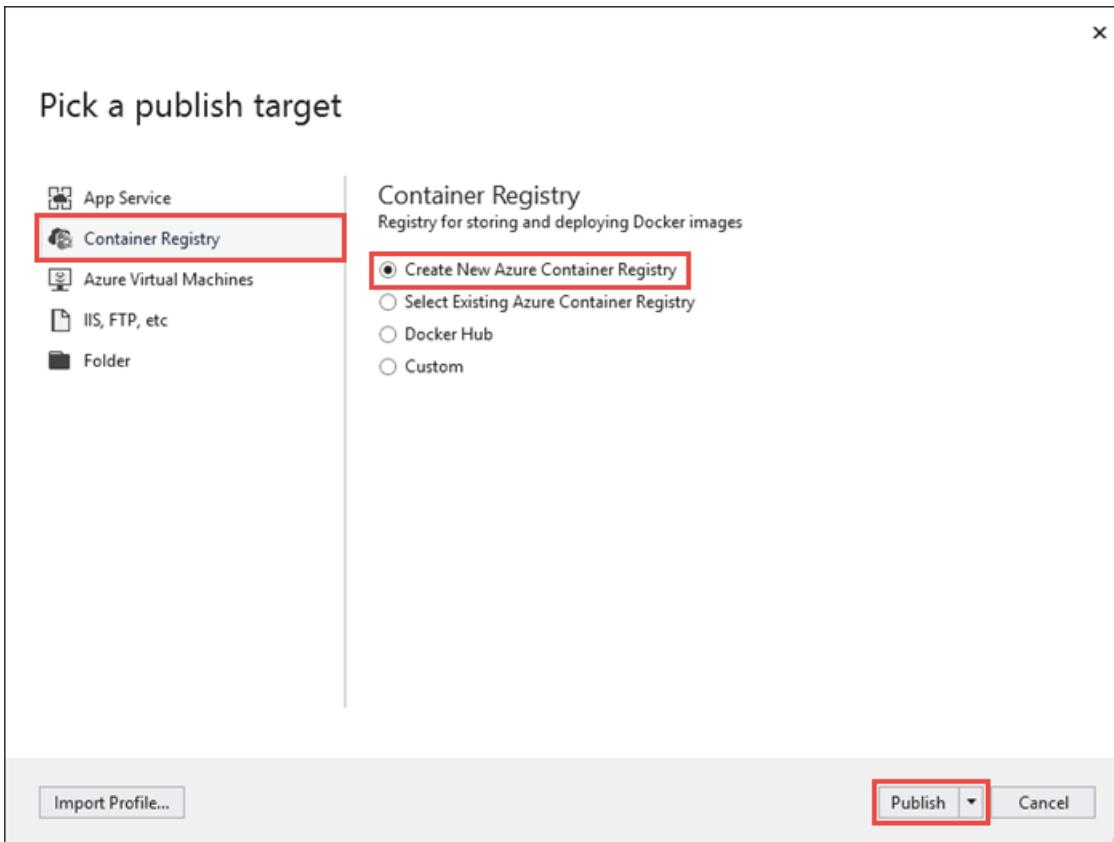
Open publish wizard

In the Solution Explorer, right-click the **CustomFontSample** project and select **Publish**.



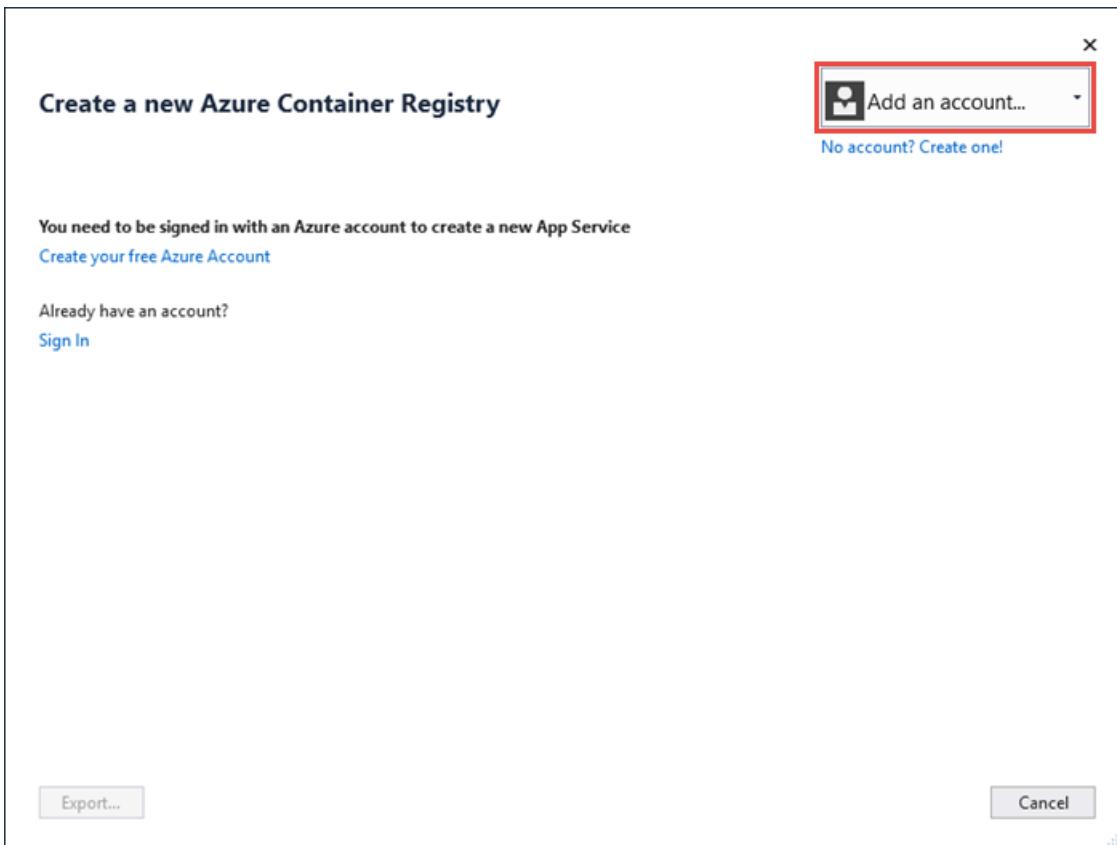
Create registry and publish

In the publish wizard, select **Container Registry > Create New Azure Container Registry > Publish**.



Sign in with Azure account

In the **Create a new Azure Container Registry** dialog, select **Add an account**, and sign in to your Azure subscription. If you're already signed in, select the account containing the desired subscription from the dropdown.



Configure the registry

Configure the new container registry based on the suggested values in the following table. When finished, click **Create**.

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
DNS Prefix	Keep the generated registry name, or change it to another unique name.	
Resource Group	Click New , type myResourceGroup , and click OK .	
SKU	Basic	Pricing tiers
Registry Location	West Europe	

Create a new Azure Container Registry

DNS Prefix
CustomFontSample20180914115836

Subscription
Visual Studio Ultimate with MSDN

Resource Group
myResourceGroup* [New...](#)

SKU
Basic

Registry Location
West Europe

Clicking the Create button will create the following Azure resources
Azure Container Registry - CustomFontSample20180914115836

Export... Create Cancel

A terminal window is opened and displays the image deployment progress. Wait for the deployment to complete.

Sign in to Azure

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

Create a web app

From the left menu, select **Create a resource > Web > Web App for Containers**.

Configure app basics

In the **Basics** tab, configure the settings according to the following table, then click **Next: Docker**.

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
Subscription	Make sure the correct subscription is selected.	
Resource Group	Select Create new , type myResourceGroup , and click OK .	
Name	Type a unique name.	The URL of the web app is <code>http://<app-name>.azurewebsites.net</code> , where <code><app-name></code> is your app name.
Publish	Docker container	
Operating System	Windows	
Region	West Europe	

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
Windows Plan	Select Create new , type myAppServicePlan , and click OK .	

Your **Basics** tab should look like this:

Basics Docker Monitoring Tags Review + create

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

Project Details

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

Subscription * ⓘ Visual Studio Ultimate with MSDN

Resource Group * ⓘ (New) myResourceGroup [Create new](#)

Instance Details

Name * custom-font-app .azurewebsites.net

Publish * [Code](#) **Docker Container**

Operating System * [Linux](#) **Windows**

Region * West Europe ⓘ Can't find your App Service Plan, try a different region.

App Service Plan

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

Windows Plan (West Europe) * ⓘ (New) myAppServicePlan [Create new](#)

Sku and size * **Premium Container PC2**
320 total ACU, 8 GB memory [Change size](#)

[Review + create](#) < Previous Next : Docker >

Configure Windows container

In the **Docker** tab, configure your custom Windows container as shown in the following table, and select **Review + create**.

SETTING	SUGGESTED VALUE
Image Source	Azure Container Register
Registry	Select the registry you created earlier.

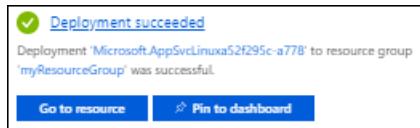
SETTING	SUGGESTED VALUE
Image	customfontsample
Tag	latest

Complete app creation

Click **Create** and wait for Azure to create the required resources.

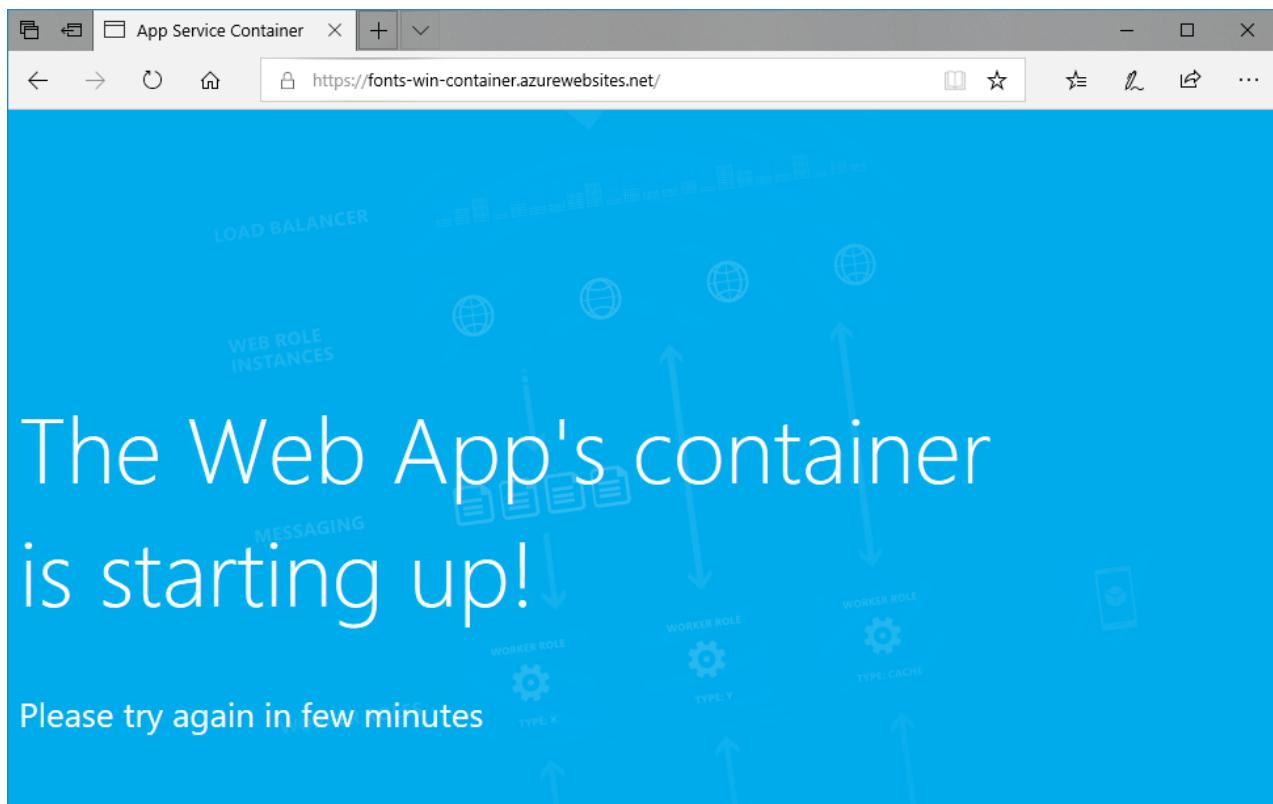
Browse to the web app

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

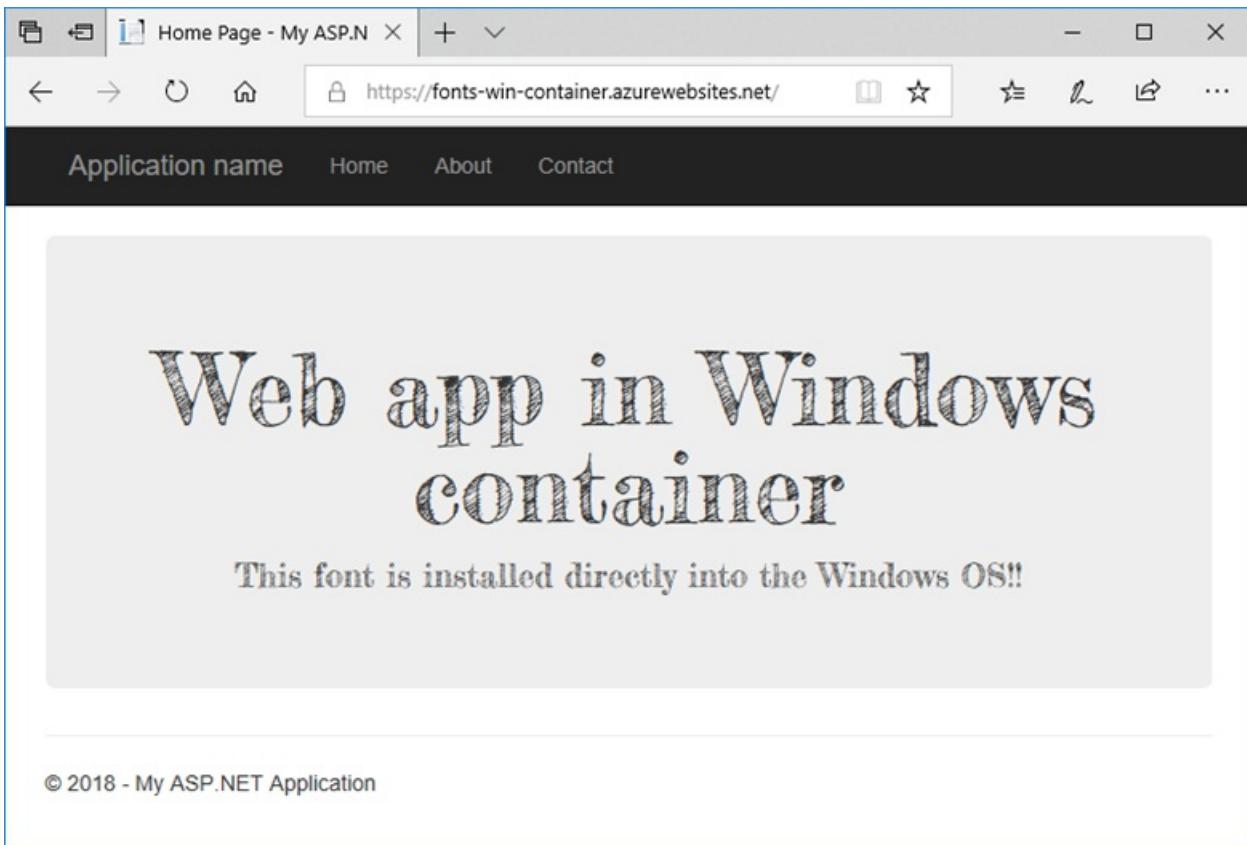


1. Click **Go to resource**.
2. In the app page, click the link under **URL**.

A new browser page is opened to the following page:



Wait a few minutes and try again, until you get the homepage with the beautiful font you expect:



Congratulations! You've migrated an ASP.NET application to Azure App Service in a Windows container.

See container start-up logs

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

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

The streamed logs looks like this:

```
14/09/2018 23:16:19.889 INFO - Site: fonts-win-container - Creating container for image: customfontsample20180914115836.azurecr.io/customfontsample:latest.
14/09/2018 23:16:19.928 INFO - Site: fonts-win-container - Create container for image: customfontsample20180914115836.azurecr.io/customfontsample:latest succeeded. Container Id 329ecfedbe370f1d99857da7352a7633366b878607994ff1334461e44e6f5418
14/09/2018 23:17:23.405 INFO - Site: fonts-win-container - Start container succeeded. Container: 329ecfedbe370f1d99857da7352a7633366b878607994ff1334461e44e6f5418
14/09/2018 23:17:28.637 INFO - Site: fonts-win-container - Container ready
14/09/2018 23:17:28.637 INFO - Site: fonts-win-container - Configuring container
14/09/2018 23:18:03.823 INFO - Site: fonts-win-container - Container ready
14/09/2018 23:18:03.823 INFO - Site: fonts-win-container - Container start-up and configuration completed successfully
```

Tutorial: Secure Azure SQL Database connection from App Service using a managed identity

2/21/2020 • 10 minutes to read • [Edit Online](#)

App Service provides a highly scalable, self-patching web hosting service in Azure. It also provides a [managed identity](#) for your app, which is a turn-key solution for securing access to [Azure SQL Database](#) and other Azure services. Managed identities in App Service make your app more secure by eliminating secrets from your app, such as credentials in the connection strings. In this tutorial, you will add managed identity to the sample web app you built in one of the following tutorials:

- [Tutorial: Build an ASP.NET app in Azure with SQL Database](#)
- [Tutorial: Build an ASP.NET Core and SQL Database app in Azure App Service](#)

When you're finished, your sample app will connect to SQL Database securely without the need of username and passwords.

NOTE

The steps covered in this tutorial support the following versions:

- .NET Framework 4.7.2
- .NET Core 2.2

What you will learn:

- Enable managed identities
- Grant SQL Database access to the managed identity
- Configure Entity Framework to use Azure AD authentication with SQL Database
- Connect to SQL Database from Visual Studio using Azure AD authentication

NOTE

Azure AD authentication is *different* from [Integrated Windows authentication](#) in on-premises Active Directory (AD DS). AD DS and Azure AD use completely different authentication protocols. For more information, see [Azure AD Domain Services documentation](#).

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

Prerequisites

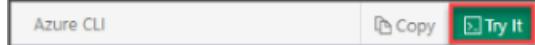
This article continues where you left off in [Tutorial: Build an ASP.NET app in Azure with SQL Database](#) or [Tutorial: Build an ASP.NET Core and SQL Database app in Azure App Service](#). If you haven't already, follow one of the two tutorials first. Alternatively, you can adapt the steps for your own .NET app with SQL Database.

To debug your app using SQL Database as the back end, make sure that you've allowed client connection from your computer. If not, add the client IP by following the steps at [Manage server-level IP firewall rules using the Azure portal](#).

Use Azure Cloud Shell

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

To start Azure Cloud Shell:

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

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

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

Grant database access to Azure AD user

First enable Azure AD authentication to SQL Database by assigning an Azure AD user as the Active Directory admin of the SQL Database server. This user is different from the Microsoft account you used to sign up for your Azure subscription. It must be a user that you created, imported, synced, or invited into Azure AD. For more information on allowed Azure AD users, see [Azure AD features and limitations in SQL Database](#).

If your Azure AD tenant doesn't have a user yet, create one by following the steps at [Add or delete users using Azure Active Directory](#).

Find the object ID of the Azure AD user using the `az ad user list` and replace <user-principal-name>. The result is saved to a variable.

```
azureaduser=$(az ad user list --filter "userPrincipalName eq '<user-principal-name>'" --query [].objectId --output tsv)
```

TIP

To see the list of all user principal names in Azure AD, run `az ad user list --query [].userPrincipalName`.

Add this Azure AD user as an Active Directory admin using `az sql server ad-admin create` command in the Cloud Shell. In the following command, replace <server-name> with the SQL Database server name (without the `.database.windows.net` suffix).

```
az sql server ad-admin create --resource-group myResourceGroup --server-name <server-name> --display-name ADMIN --object-id $azureaduser
```

For more information on adding an Active Directory admin, see [Provision an Azure Active Directory administrator for your Azure SQL Database Server](#)

Set up Visual Studio

Windows

Visual Studio for Windows is integrated with Azure AD authentication. To enable development and debugging in Visual Studio, add your Azure AD user in Visual Studio by selecting **File > Account Settings** from the menu, and click **Add an account**.

To set the Azure AD user for Azure service authentication, select **Tools > Options** from the menu, then select **Azure Service Authentication > Account Selection**. Select the Azure AD user you added and click **OK**.

You're now ready to develop and debug your app with the SQL Database as the back end, using Azure AD authentication.

MacOS

Visual Studio for Mac is not integrated with Azure AD authentication. However, the [Microsoft.Azure.Services.AppAuthentication](#) library that you will use later can use tokens from Azure CLI. To enable development and debugging in Visual Studio, first you need to [install Azure CLI](#) on your local machine.

Once Azure CLI is installed on your local machine, sign in to Azure CLI with the following command using your Azure AD user:

```
az login --allow-no-subscriptions
```

You're now ready to develop and debug your app with the SQL Database as the back end, using Azure AD authentication.

Modify your project

The steps you follow for your project depends on whether it's an ASP.NET project or an ASP.NET Core project.

- [Modify ASP.NET](#)
- [Modify ASP.NET Core](#)

Modify ASP.NET

In Visual Studio, open the Package Manager Console and add the NuGet package [Microsoft.Azure.Services.AppAuthentication](#):

```
Install-Package Microsoft.Azure.Services.AppAuthentication -Version 1.3.1
```

In *Web.config*, working from the top of the file and make the following changes:

- In `<configSections>`, add the following section declaration in it:

```
<section name="SqlAuthenticationProviders"
type="System.Data.SqlClient.SqlAuthenticationProviderConfigurationSection, System.Data, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
```

- below the closing `</configSections>` tag, add the following XML code for `<SqlAuthenticationProviders>`.

```

<SqlAuthenticationProviders>
  <providers>
    <add name="Active Directory Interactive"
      type="Microsoft.Azure.Services.AppAuthentication.SqlAppAuthenticationProvider,
      Microsoft.Azure.Services.AppAuthentication" />
  </providers>
</SqlAuthenticationProviders>

```

- Find the connection string called `MyDbConnection` and replace its `connectionString` value with
`"server=tcp:<server-name>.database.windows.net;database=<db-name>;UID=AnyString;Authentication=Active Directory Interactive"`
 Replace `<server-name>` and `<db-name>` with your server name and database name.

That's every thing you need to connect to SQL Database. When debugging in Visual Studio, your code uses the Azure AD user you configured in [Set up Visual Studio](#). You'll set up the SQL Database server later to allow connection from the managed identity of your App Service app.

Type `ctrl+F5` to run the app again. The same CRUD app in your browser is now connecting to the Azure SQL Database directly, using Azure AD authentication. This setup lets you run database migrations from Visual Studio.

Modify ASP.NET Core

In Visual Studio, open the Package Manager Console and add the NuGet package `Microsoft.Azure.Services.AppAuthentication`:

```
Install-Package Microsoft.Azure.Services.AppAuthentication -Version 1.3.1
```

In the [ASP.NET Core and SQL Database tutorial](#), the `MyDbConnection` connection string isn't used at all because the local development environment uses a Sqlite database file, and the Azure production environment uses a connection string from App Service. With Active Directory authentication, you want both environments to use the same connection string. In `appsettings.json`, replace the value of the `MyDbConnection` connection string with:

```
"Server=tcp:<server-name>.database.windows.net,1433;Database=<database-name>;"
```

In `Startup.cs`, remove the code section that you added before:

```

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

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

```

And replace it with the following code:

```

services.AddDbContext<MyDatabaseContext>(options => {
    options.UseSqlServer(Configuration.GetConnectionString("MyDbConnection"));
});

```

Next, you supply the Entity Framework database context with the access token for the SQL Database. In

`Data\MyDatabaseContext.cs`, add the following code inside the curly braces of the empty `MyDatabaseContext (DbContextOptions<MyDatabaseContext> options)` constructor:

```
var conn = (System.Data.SqlClient.SqlConnection)Database.GetDbConnection();
conn.AccessToken = (new
Microsoft.Azure.Services.AppAuthentication.AzureServiceTokenProvider()).GetAccessTokenAsync("https://database.
windows.net/").Result;
```

NOTE

This demonstration code is synchronous for clarity and simplicity.

That's every thing you need to connect to SQL Database. When debugging in Visual Studio, your code uses the Azure AD user you configured in [Set up Visual Studio](#). You'll set up the SQL Database server later to allow connection from the managed identity of your App Service app. The `AzureServiceTokenProvider` class caches the token in memory and retrieves it from Azure AD just before expiration. You don't need any custom code to refresh the token.

TIP

If the Azure AD user you configured has access to multiple tenants, call

```
GetAccessTokenAsync("https://database.windows.net/", tenantid)
```

 with the desired tenant ID to retrieve the proper access token.

Type `ctrl+F5` to run the app again. The same CRUD app in your browser is now connecting to the Azure SQL Database directly, using Azure AD authentication. This setup lets you run database migrations from Visual Studio.

Use managed identity connectivity

Next, you configure your App Service app to connect to SQL Database with a system-assigned managed identity.

Enable managed identity on app

To enable a managed identity for your Azure app, use the `az webapp identity assign` command in the Cloud Shell. In the following command, replace `<app-name>`.

```
az webapp identity assign --resource-group myResourceGroup --name <app-name>
```

Here's an example of the output:

```
{
  "additionalProperties": {},
  "principalId": "21dfa71c-9e6f-4d17-9e90-1d28801c9735",
  "tenantId": "72f988bf-86f1-41af-91ab-2d7cd011db47",
  "type": "SystemAssigned"
}
```

Grant permissions to managed identity

NOTE

If you want, you can add the identity to an [Azure AD group](#), then grant SQL Database access to the Azure AD group instead of the identity. For example, the following commands add the managed identity from the previous step to a new group called *myAzureSQLDBAccessGroup*:

```
groupid=$(az ad group create --display-name myAzureSQLDBAccessGroup --mail-nickname myAzureSQLDBAccessGroup --query objectId --output tsv)
msiobjectid=$(az webapp identity show --resource-group myResourceGroup --name <app-name> --query principalId --output tsv)
az ad group member add --group $groupid --member-id $msiobjectid
az ad group member list -g $groupid
```

In the Cloud Shell, sign in to SQL Database by using the SQLCMD command. Replace *<server-name>* with your SQL Database server name, *<db-name>* with the database name your app uses, and *<aad-user-name>* and *<aad-password>* with your Azure AD user's credentials.

```
sqlcmd -S <server-name>.database.windows.net -d <db-name> -U <aad-user-name> -P "<aad-password>" -G -l 30
```

In the SQL prompt for the database you want, run the following commands to add the Azure AD group and grant the permissions your app needs. For example,

```
CREATE USER [<identity-name>] FROM EXTERNAL PROVIDER;
ALTER ROLE db_datareader ADD MEMBER [<identity-name>];
ALTER ROLE db_datawriter ADD MEMBER [<identity-name>];
ALTER ROLE db_ddladmin ADD MEMBER [<identity-name>];
GO
```

<identity-name> is the name of the managed identity in Azure AD. Since it's system-assigned, it's always the same as the name of your App Service app. To grant permissions for an Azure AD group, use the group's display name instead (for example, *myAzureSQLDBAccessGroup*).

Type `EXIT` to return to the Cloud Shell prompt.

Modify connection string

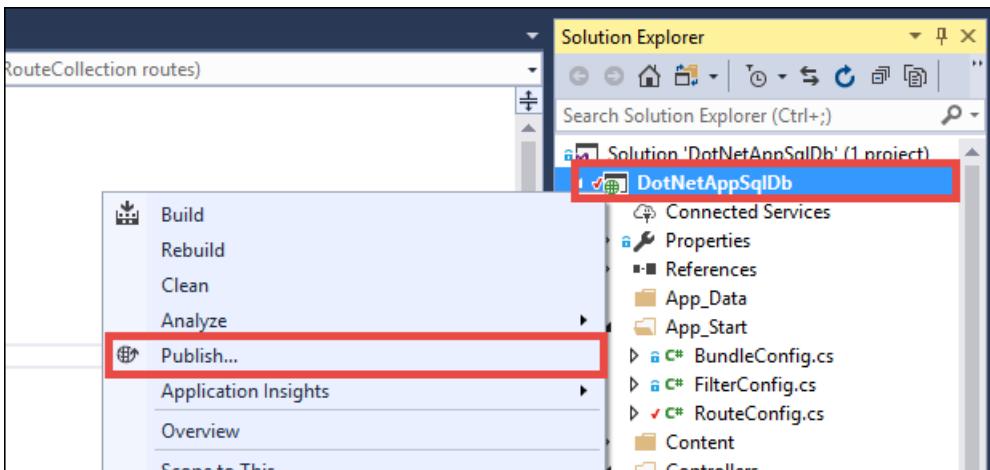
Remember that the same changes you made in *Web.config* or *appsettings.json* works with the managed identity, so the only thing to do is to remove the existing connection string in App Service, which Visual Studio created deploying your app the first time. Use the following command, but replace *<app-name>* with the name of your app.

```
az webapp config connection-string delete --resource-group myResourceGroup --name <app-name> --setting-names MyDbConnection
```

Publish your changes

All that's left now is to publish your changes to Azure.

If you came from [Tutorial: Build an ASP.NET app in Azure with SQL Database](#), publish your changes in Visual Studio. In the **Solution Explorer**, right-click your **DotNetAppSqlDb** project and select **Publish**.



In the publish page, click **Publish**.

If you came from [Tutorial: Build an ASP.NET Core and SQL Database app in Azure App Service](#), publish your changes using Git, with the following commands:

```
git commit -am "configure managed identity"  
git push azure master
```

When the new webpage shows your to-do list, your app is connecting to the database using the managed identity.

A screenshot of a web browser window. The address bar shows 'Index - My ASP.NET App' and the URL 'dotnetappsqldb1234.azurewebsites.net'. The page title is 'My TodoList App'. Below it, the word 'Todos' is displayed. There is a 'Create New' button. A table lists three items: 'Deploy app to Azure' (Created Date: 2017-06-01, Done: checked, links: Edit | Details | Delete); 'Walk dog' (Created Date: 2017-06-03, Done: unchecked, links: Edit | Details | Delete); and 'Feed cat' (Created Date: 2017-06-04, Done: unchecked, links: Edit | Details | Delete). At the bottom of the page, there is a copyright notice: '© 2017 - My ASP.NET Application'.

You should now be able to edit the to-do list as before.

Clean up resources

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

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

This command may take a minute to run.

Next steps

What you learned:

- Enable managed identities
- Grant SQL Database access to the managed identity
- Configure Entity Framework to use Azure AD authentication with SQL Database
- Connect to SQL Database from Visual Studio using Azure AD authentication

Advance to the next tutorial to learn how to map a custom DNS name to your web app.

[Map an existing custom DNS name to Azure App Service](#)

Tutorial: Host a RESTful API with CORS in Azure App Service

2/20/2020 • 9 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. In addition, App Service has built-in support for [Cross-Origin Resource Sharing \(CORS\)](#) for RESTful APIs. This tutorial shows how to deploy an ASP.NET Core API app to App Service with CORS support. You configure the app using command-line tools and deploy the app using Git.

In this tutorial, you learn how to:

- Create App Service resources using Azure CLI
- Deploy a RESTful API to Azure using Git
- Enable App Service CORS support

You can follow the steps in this tutorial on macOS, Linux, Windows.

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

Prerequisites

To complete this tutorial:

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

Create local ASP.NET Core app

In this step, you set up the local ASP.NET Core project. App Service supports the same workflow for APIs written in other languages.

Clone the sample application

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/dotnet-core-api
```

This repository contains an app that's created based on the following tutorial: [ASP.NET Core Web API help pages using Swagger](#). It uses a Swagger generator to serve the [Swagger UI](#) and the Swagger JSON endpoint.

Run the application

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

```
cd dotnet-core-api  
dotnet restore  
dotnet run
```

Navigate to `http://localhost:5000/swagger` in a browser to play with the Swagger UI.

Navigate to <http://localhost:5000/api/todo> and see a list of ToDo JSON items.

Navigate to <http://localhost:5000> and play with the browser app. Later, you will point the browser app to a remote API in App Service to test CORS functionality. Code for the browser app is found in the repository's `wwwroot` directory.

To stop ASP.NET Core at any time, press `Ctrl+C` in the terminal.

Use Azure Cloud Shell

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

To start Azure Cloud Shell:

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

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

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

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

Deploy app to Azure

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

Configure local git deployment

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

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

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

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

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

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

Create a resource group

A [resource group](#) is a logical container into which Azure resources 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 in **Free** tier, run the [az appservice list-locations --sku FREE](#) command.

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

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

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

Create an App Service plan

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

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

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

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

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

Create a web app

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

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --deployment-local-git
```

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

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

NOTE

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

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

Push to Azure from Git

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: .  
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

Navigate to http://<app_name>.azurewebsites.net/swagger in a browser and play with the Swagger UI.

The screenshot shows the Swagger UI interface for a deployed API. At the top, there's a green header bar with the 'swagger' logo, a dropdown menu 'Select a spec' set to 'My API V1', and a 'Authorize' button with a lock icon. Below the header, the title 'My API v1' is displayed, followed by a link to the Swagger JSON file: '/swagger/v1/swagger.json'. The main content area is titled 'Todo' and contains four API operations:

- A blue 'GET' button next to the URL '/api/Todo'.
- A green 'POST' button next to the URL '/api/Todo'.
- A blue 'GET' button next to the URL '/api/Todo/{id}'.
- An orange 'PUT' button next to the URL '/api/Todo/{id}'.

Navigate to `http://<app_name>.azurewebsites.net/swagger/v1/swagger.json` to see the `swagger.json` for your deployed API.

Navigate to `http://<app_name>.azurewebsites.net/api/todo` to see your deployed API working.

Add CORS functionality

Next, you enable the built-in CORS support in App Service for your API.

Test CORS in sample app

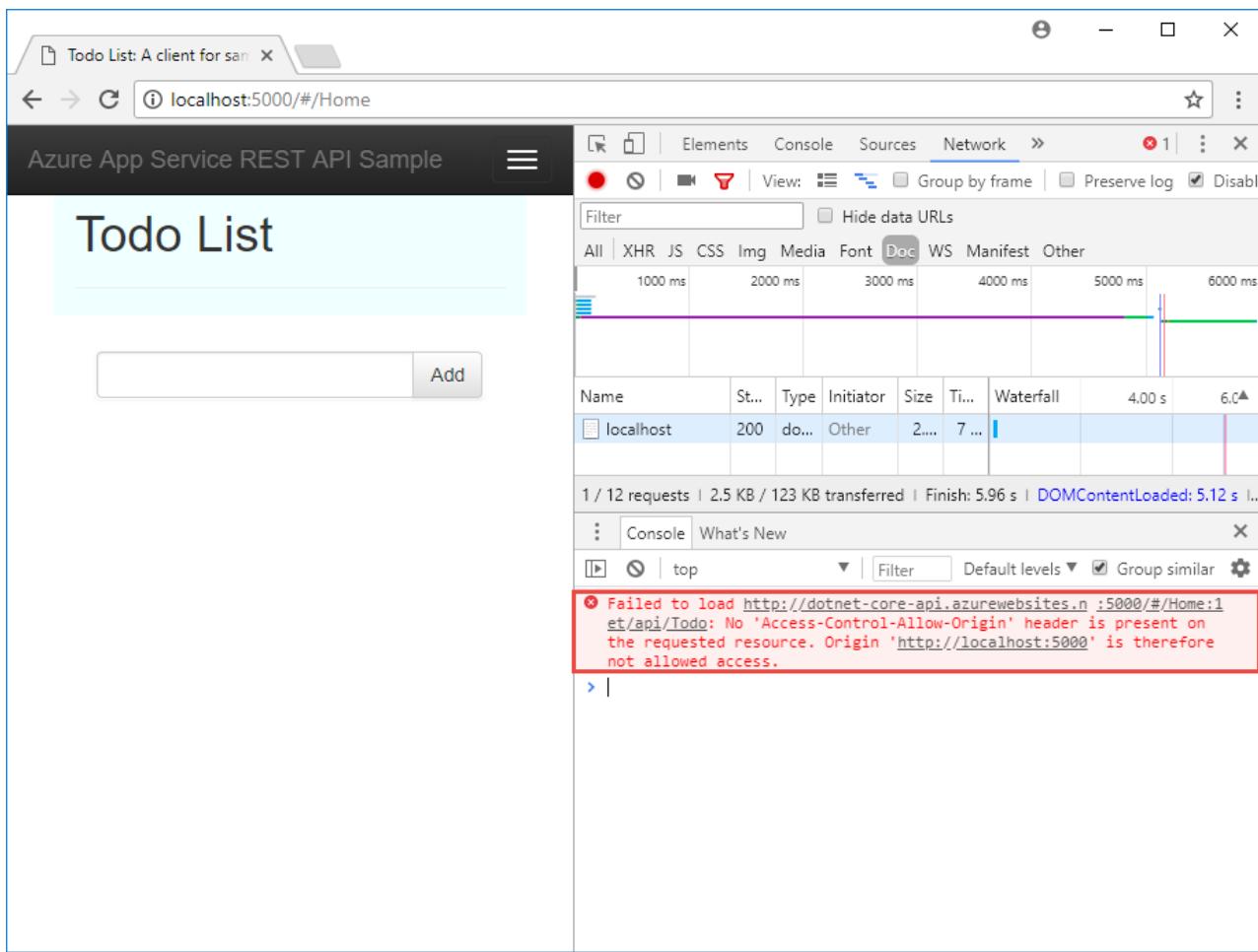
In your local repository, open `wwwroot/index.html`.

In Line 51, set the `apiEndpoint` variable to the URL of your deployed API (`http://<app_name>.azurewebsites.net`). Replace `<appname>` with your app name in App Service.

In your local terminal window, run the sample app again.

```
dotnet run
```

Navigate to the browser app at `http://localhost:5000`. Open the developer tools window in your browser (`Ctrl + Shift + i` in Chrome for Windows) and inspect the **Console** tab. You should now see the error message, 'No 'Access-Control-Allow-Origin' header is present on the requested resource'.



Because of the domain mismatch between the browser app (`http://localhost:5000`) and remote resource (`http://<app_name>.azurewebsites.net`), and the fact that your API in App Service is not sending the `Access-Control-Allow-Origin` header, your browser has prevented cross-domain content from loading in your browser app.

In production, your browser app would have a public URL instead of the localhost URL, but the way to enable CORS to a localhost URL is the same as a public URL.

Enable CORS

In the Cloud Shell, enable CORS to your client's URL by using the `az resource update` command. Replace the `<appname>` placeholder.

```
az resource update --name web --resource-group myResourceGroup --namespace Microsoft.Web --resource-type config --parent sites/<app_name> --set properties.cors.allowedOrigins="['http://localhost:5000']" --api-version 2015-06-01
```

You can set more than one client URL in `properties.cors.allowedOrigins` (`"['URL1','URL2',...]"`). You can also enable all client URLs with `"['*']"`.

NOTE

If your app requires credentials such as cookies or authentication tokens to be sent, the browser may require the `ACCESS-CONTROL-ALLOW-CREDENTIALS` header on the response. To enable this in App Service, set `properties.cors.supportCredentials` to `true` in your CORS config. This cannot be enabled when `allowedOrigins` includes `'*'`.

Test CORS again

Refresh the browser app at <http://localhost:5000>. The error message in the **Console** window is now gone, and you can see the data from the deployed API and interact with it. Your remote API now supports CORS to your browser app running locally.

The screenshot shows a browser window with the title "Todo List: A client for san". The address bar says "localhost:5000/#/Home". The main content area displays a "Todo List" application with a single item: "Item1" with "Edit | Delete" links. To the right of the browser is the Microsoft Edge DevTools interface. The "Network" tab is selected, showing a timeline of requests. One request to "localhost" is listed, showing a status of 200, type "do...", initiator "Other", size "2....", and time "7...". The total transfer is "2.5 KB / 123 KB transferred" and the finish time is "3.30 s". The "Console" tab shows the message "DOMContentLoaded: 249 ms...".

Congratulations, you're running an API in Azure App Service with CORS support.

App Service CORS vs. your CORS

You can use your own CORS utilities instead of App Service CORS for more flexibility. For example, you may want to specify different allowed origins for different routes or methods. Since App Service CORS lets you specify one set of accepted origins for all API routes and methods, you would want to use your own CORS code. See how ASP.NET Core does it at [Enabling Cross-Origin Requests \(CORS\)](#).

NOTE

Don't try to use App Service CORS and your own CORS code together. When used together, App Service CORS takes precedence and your own CORS code has no effect.

Clean up resources

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

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

This command may take a minute to run.

Next steps

What you learned:

- Create App Service resources using Azure CLI
- Deploy a RESTful API to Azure using Git
- Enable App Service CORS support

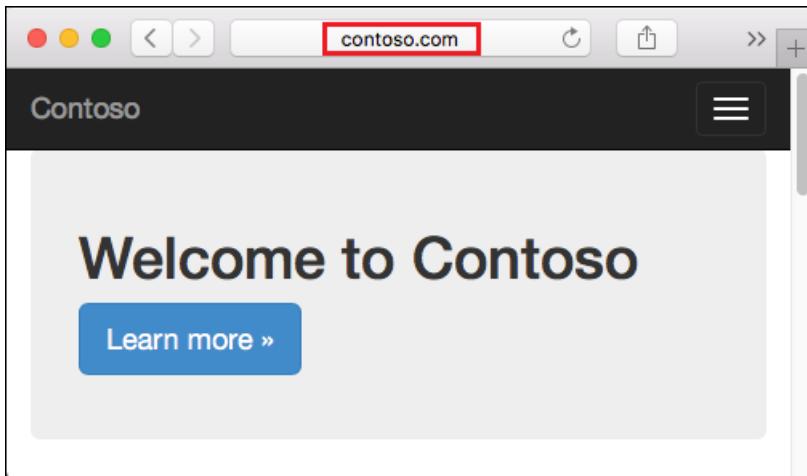
Advance to the next tutorial to learn how to authenticate and authorize users.

[Tutorial: Authenticate and authorize users end-to-end](#)

Tutorial: 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

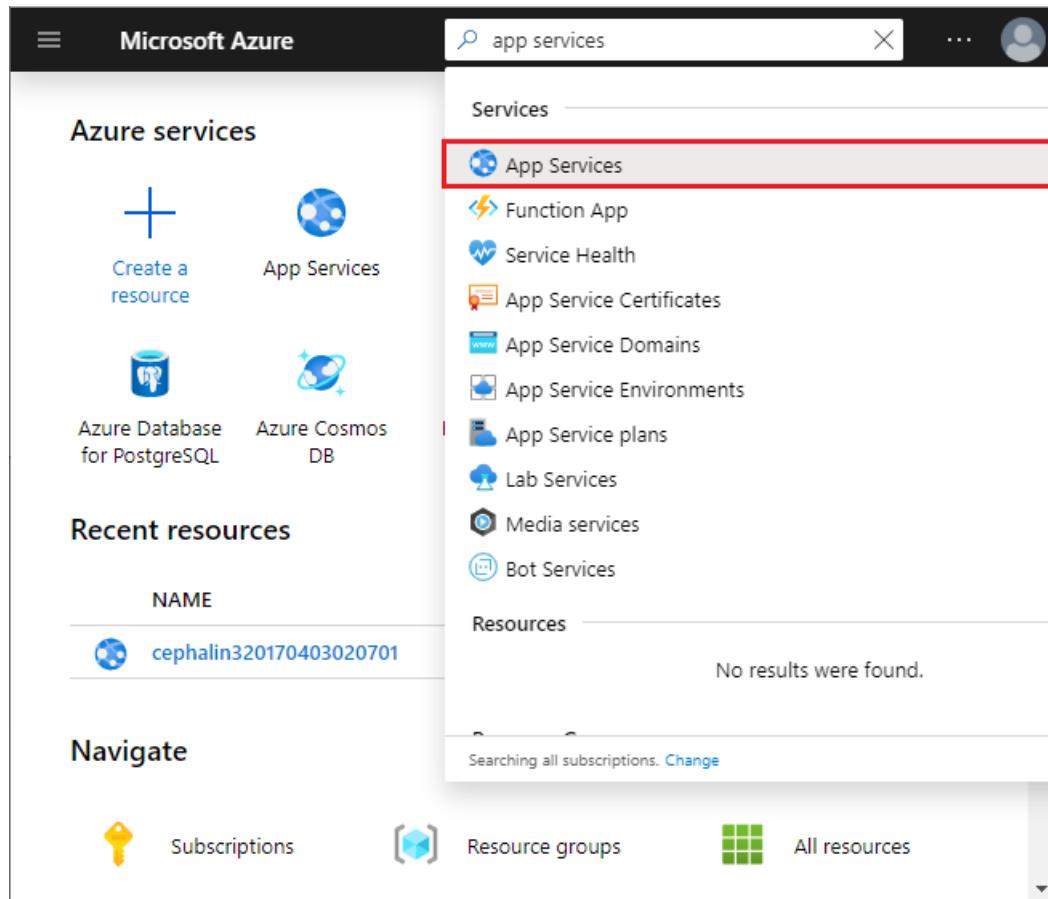
App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure virtual machines as other App Service apps. Some apps might belong to other customers. These tiers are intended to be used only for development and testing purposes.

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" such as "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". To the right of the sidebar, the main content area displays a list of services under the heading "Services". The "App Services" item is highlighted with a red box. Other listed services include "Function App", "Service Health", "App Service Certificates", "App Service Domains", "App Service Environments", "App Service plans", "Lab Services", "Media services", and "Bot Services". Below this list, there's a section titled "Resources" with the message "No results were found." At the bottom of the page, there are navigation 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

+ Add Refresh Start 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

<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

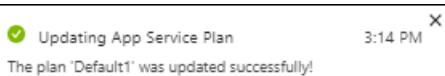
Memory per instance available to run applications deployed and running in...

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

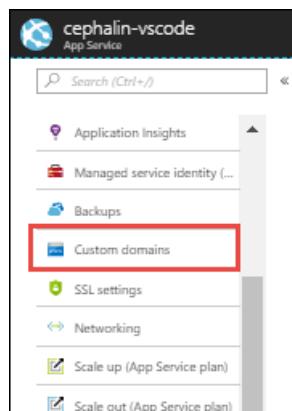
Last updated 6/18/2018 3:43 PM

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

ADD

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**.

A screenshot of the 'Custom Domains' page in the Azure portal. It shows a table with one row: a green checkmark under 'SSL STATE', the text 'Secure' under 'ASSIGNED CUSTOM DOMAINS', and 'my-demo-app.azurewebsites.net' under 'SSL BINDING'. At the top, there is a 'Status Filter' dropdown set to 'All (1)', and below it, a 'Add custom domain' button with a plus sign icon, which is highlighted with a red box.

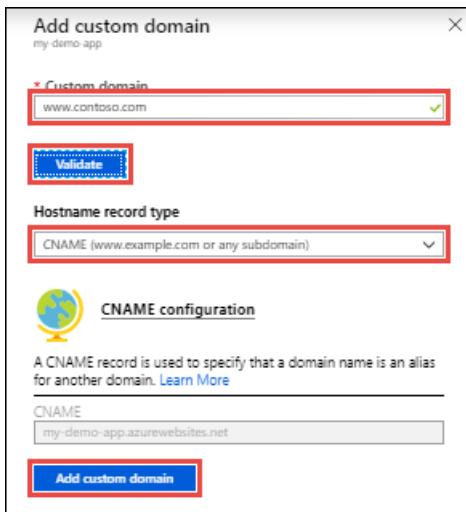
Type the fully qualified domain name that you added a CNAME record for, such as `www.contoso.com`.

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**.



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.

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.

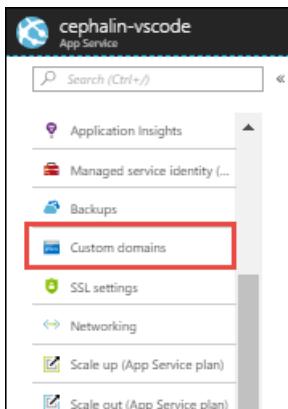
Map an A record

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.

Custom Hostnames

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

IP address: 40.118.102.46

HTTPS Only: Off On

[Add hostname](#)

HOSTNAMES ASSIGNED TO SITE SSL BINDING

cephalin-vscode.azurewebsites.net

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				
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 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 Copy the app's IP address
TXT	@	<code><app_name>.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 Copy the app's IP address
TXT	www	<code><app_name>.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 '13.09.68.6' and 'HTTPS Only:' with a switch set to 'Off'. A large blue '+' button labeled 'Add custom domain' is prominently displayed. Below these are sections for 'Status Filter' (with 'All (1)' selected), 'SSL STATE' (with 'Secure' checked), and 'ASSIGNED CUSTOM DOMAINS' (listing 'my-demo-app.azurewebsites.net').

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 fields: 'Custom domain' (contoso.com), 'Validate' (button), 'Hostname record type' (A record (example.com)), 'A record configuration' (described as mapping the domain to an IP address), 'External IP address' (13.09.68.6), and an 'Add custom domain' button. The 'Custom domain' and 'Hostname record type' fields are highlighted with red boxes.

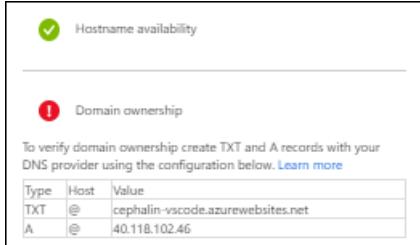
It might take some time for the new custom domain to be reflected in the app's **Custom domains** page. Try refreshing the browser to update the data.

The screenshot shows the 'Custom Domains' page again. The 'Status Filter' is now set to 'All (2)'. The table lists two domains: 'contoso.com' (status 'Not Secure') and 'my-demo-app.azurewebsites.net' (status 'Secure'). The 'contoso.com' row has an 'Add binding' link. The 'contoso.com' row is highlighted with a red box.

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:

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 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:

The screenshot shows the 'Records' section of the Azure portal. It lists a single CNAME record with the following details:

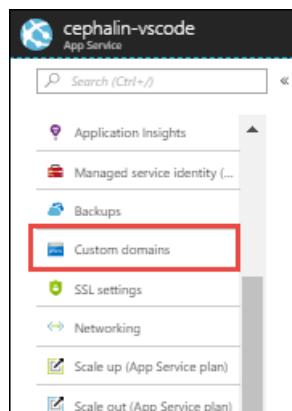
Type	Name	Value	TTL
CNAME	*	cephalin-vscode.azurewebsites.net	1 Hour

At the bottom right, there is a blue 'ADD' button with a pencil icon.

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**.

The screenshot shows the 'Custom Domains' configuration page. It includes fields for IP address (13.69.68.6), HTTPS Only (Off), and a large '+ Add custom domain' button. Below these are status filters for All (1), Not Secure (0), and Secure (1). A table at the bottom lists one assigned custom domain: my-demo-app.azurewebsites.net, which is marked as secure.

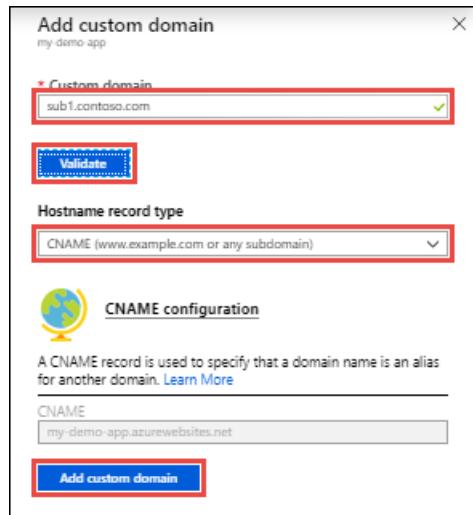
Type a fully qualified domain name that matches the wildcard domain (for example, `sub1.contoso.com`), and

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. It includes the following sections:

- Configure and manage custom domains assigned to your app**: Includes a 'Learn more' link.
- IP address:** Set to '13.69.68.6'.
- HTTPS Only:** A switch set to 'Off'.
- Add custom domain:** A button with a plus sign.
- Status Filter:** Buttons for 'All (3)', 'Not Secure (2)', and 'Secure (1)'.
- SSL STATE**, **ASSIGNED CUSTOM DOMAINS**, and **SSL BINDING** columns for the following entries:

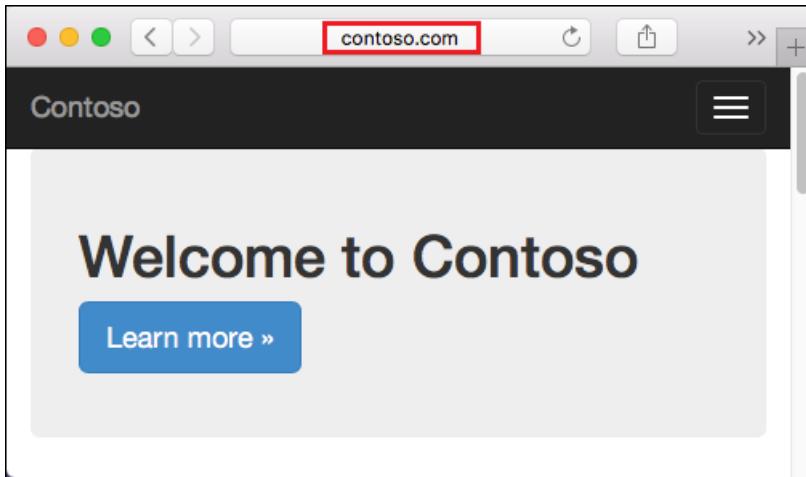
SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
! Not Secure	sub1.contoso.com	Add binding
! Not Secure	sub2.contoso.com	Add binding
✓ Secure	my-demo-app.azurewebsites.net	

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

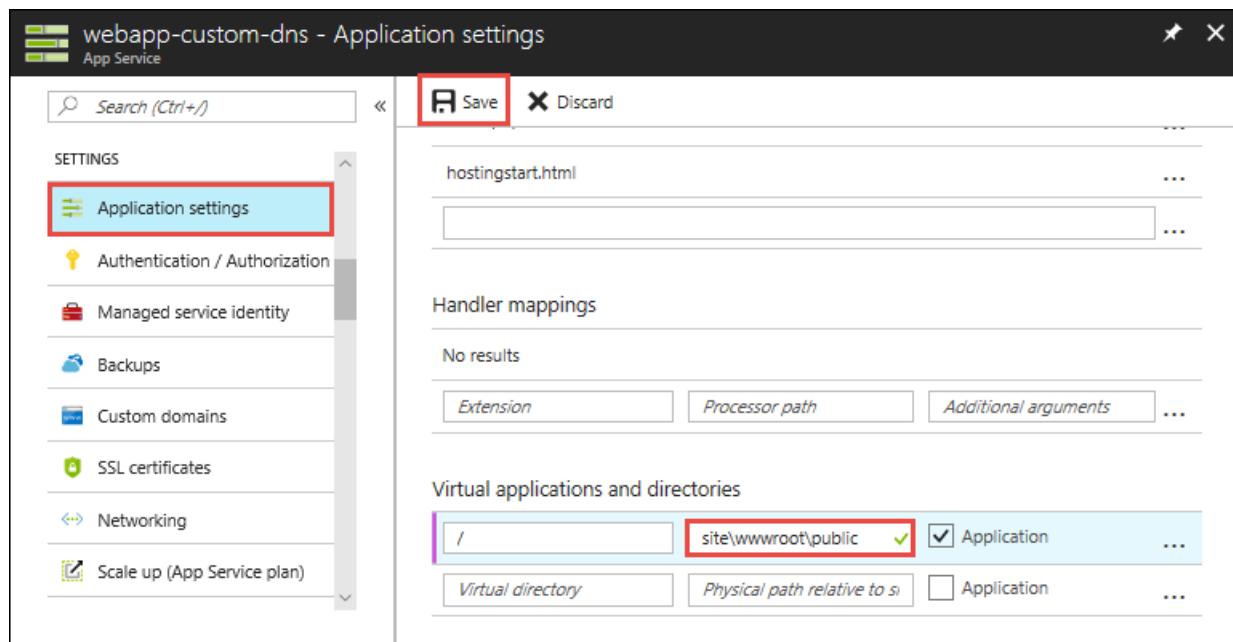
To migrate a live site and its DNS domain name to App Service with no downtime, see [Migrate an active DNS name to Azure App Service](#).

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.



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

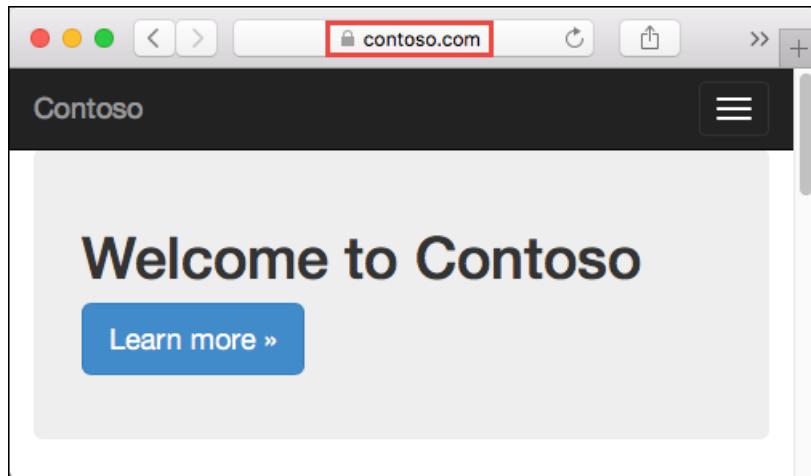
Advance to the next tutorial to learn how to bind a custom SSL certificate to a web app.

[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](#)

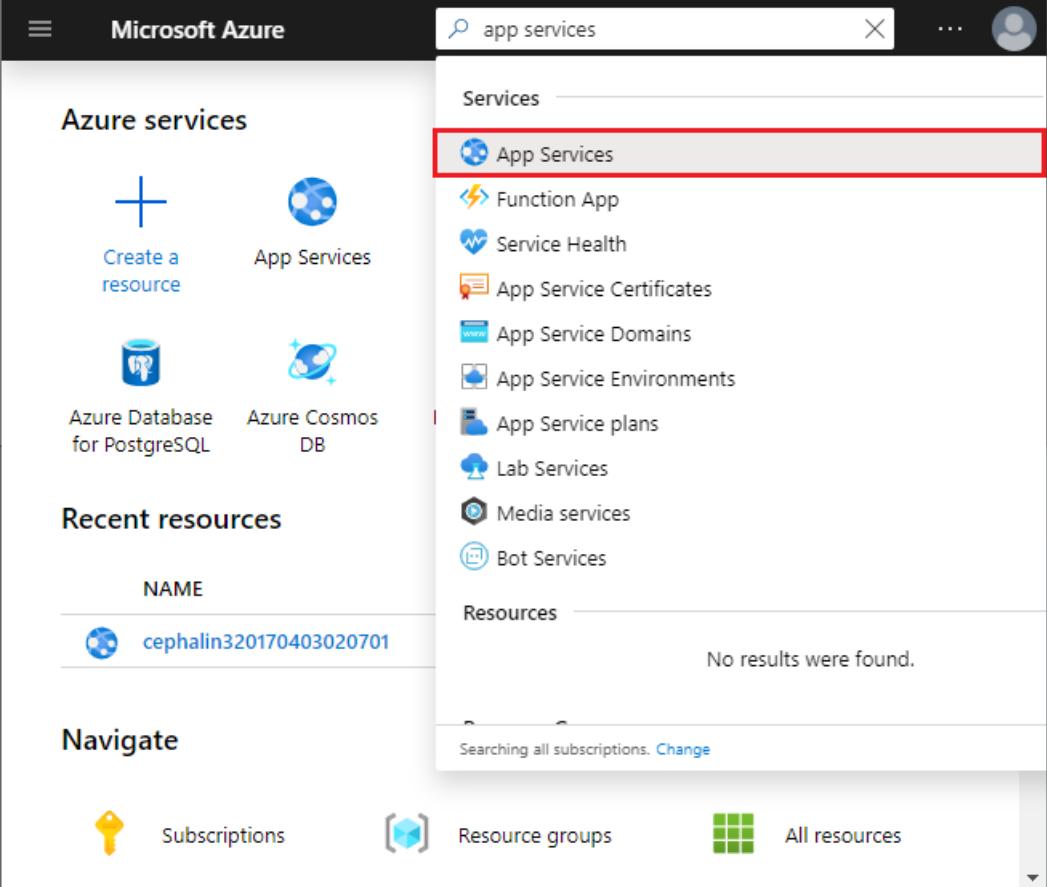
Service plan must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

Sign in to Azure

Open the [Azure portal](#).

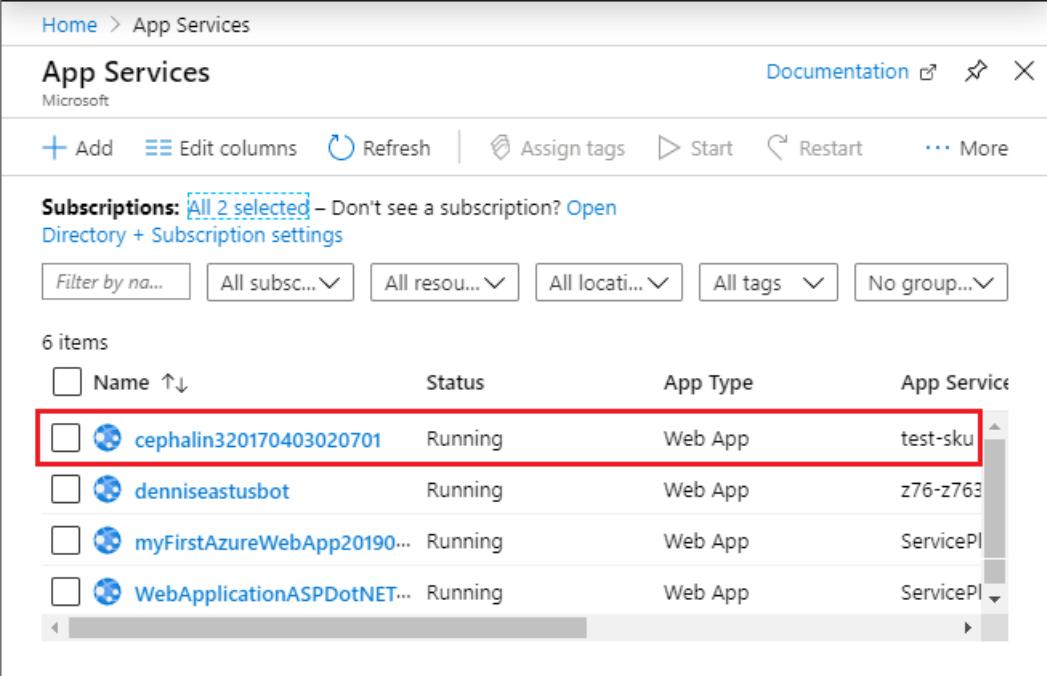
Navigate to your web app

Search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "app services". Below the search bar, the "Services" section is expanded, showing various Azure services. The "App Services" option is highlighted with a red box. Other listed services include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. To the left, there's a sidebar titled "Azure services" with options like "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". Below the sidebar is a "Recent resources" section showing a single item: "cephalin320170403020701". At the bottom, there's a "Navigate" section with links for "Subscriptions", "Resource groups", and "All resources".

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



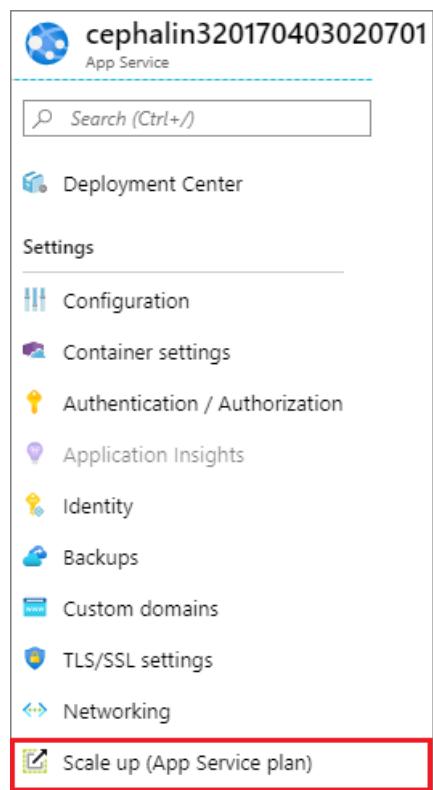
The screenshot shows the "App Services" management page. At the top, there's a header with "App Services" and "Microsoft". Below the header are buttons for "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", and "More". A message says "Subscriptions: All 2 selected - Don't see a subscription? Open Directory + Subscription settings". There are several filter buttons: "Filter by na...", "All subsc... ▾", "All resou... ▾", "All locati... ▾", "All tags ▾", and "No group... ▾". The main area shows a table with 6 items. The columns are "Name", "Status", "App Type", and "App Service". The first item, "cephalin320170403020701", is highlighted with a red box. The other items are "denniseastusbot", "myFirstAzureWebApp20190...", and "WebApplicationASPDotNET...".

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)**.



Check to make sure that your web app is not in the **F1** or **D1** tier. Your web app's current tier is highlighted by a dark blue box.



Dev / Test

For less demanding workloads



Production

For most production workloads



Isolated

Advanced networking and scale

Recommended pricing tiers

F1

Shared infrastructure
1 GB memory
60 minutes/day compute
Free

D1

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

B1

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

▼ See additional options

Included hardware

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

Azure Compute Units (ACU)



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

Memory



Memory available to run applications

Custom SSL is not supported in the **F1** or **D1** tier. If you need to scale up, follow the steps in the next section. Otherwise, close the **Scale up** page and skip the [Scale up your App Service plan](#) section.

Scale up your App Service plan

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

Click **Apply**.



Dev / Test

For less demanding workloads



Production

For most production workloads



Isolated

Advanced networking and scale

Recommended pricing tiers

F1

Shared infrastructure
1 GB memory
60 minutes/day compute
Free

D1

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

B1

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

▼ See additional options

Included features

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

**Custom domains / SSL**

Configure and purchase custom domains with SNI SSL bindings

**Manual scale**

Up to 3 instances. Subject to availability.

Included hardware

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

Azure Compute Units (ACU)

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

Memory

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

Storage

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

Apply

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



Secure a custom domain

Do the following steps:

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, start the **TLS/SSL Binding** dialog by:

- Selecting **Custom domains > Add binding**
- Selecting **TLS/SSL settings > Add TLS/SSL binding**

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

In **Custom Domain**, select the custom domain you want to add a binding for.

If your app already has a certificate for the selected custom domain, go to [Create binding](#) directly. Otherwise, keep going.

Add a certificate for custom domain

If your app has no certificate for the selected custom domain, then you have two options:

- **Upload PFX Certificate** - Follow the workflow at [Upload a private certificate](#), then select this option here.
- **Import App Service Certificate** - Follow the workflow at [Import an App Service certificate](#), then select this option here.

NOTE

You can also [Create a free certificate](#) (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"> SNI SSL - 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 Server Name Indication). IP SSL - 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 Remap A record for IP SSL. IP SSL is supported only in Production or Isolated tiers.

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

Status Filter (All (2) Not Secure (0) Secure (2))			
SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING	
Secure	contoso.com	SNI SSL	...
Secure	my-demo-app.azurewebsites.net		

NOTE

A **Secure** state in the **Custom domains** means that it is secured with a certificate, but App Service doesn't check if the certificate is self-signed or expired, for example, which can also cause browsers to show an error or warning.

Remap A record for IP SSL

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

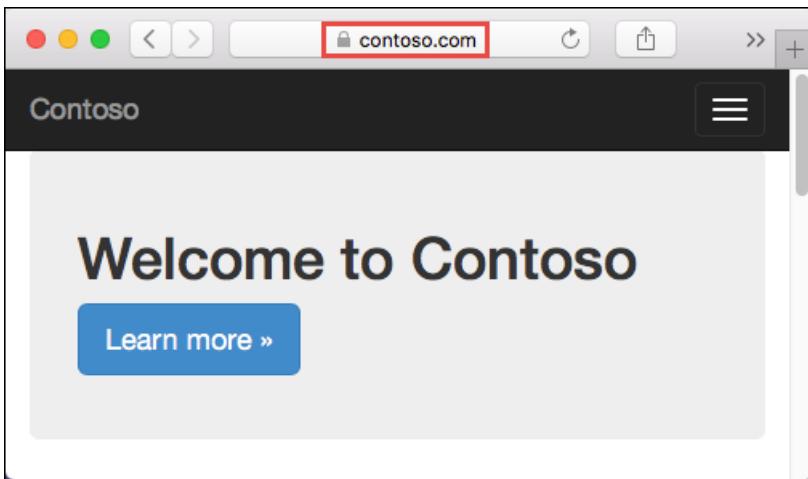
By default, your app uses a shared public IP address. When you bind a certificate with IP SSL, App Service creates a new, dedicated IP address for your app.

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**.

The screenshot shows the Azure portal's 'SSL settings' configuration for an app service. The left sidebar lists various settings like deployment slots, authentication, and backups, with 'SSL settings' currently selected. The main content area is titled 'SSL Configuration' and explains how it lets you configure TLS versions and enable HTTPS only. A large button labeled 'HTTPS Only: On' has a red box around it, indicating it's the active setting. Below this, there are tabs for 'TLS Version' (set to '> 1.0') and 'Bindings' (which describes how to specify a certificate for a specific hostname). At the bottom, there's a link to learn more about HTTPS.

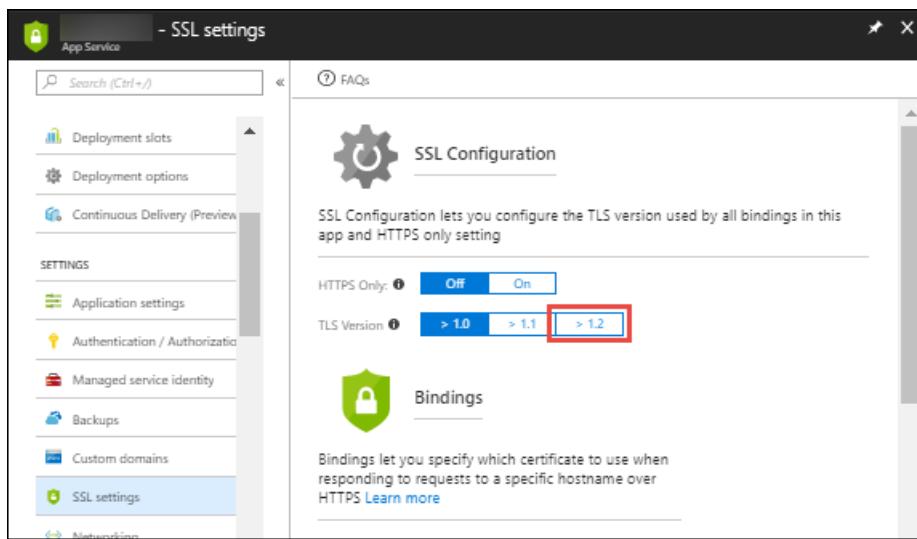
When the operation is complete, navigate to any of the HTTP URLs that point to your app. For example:

- `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

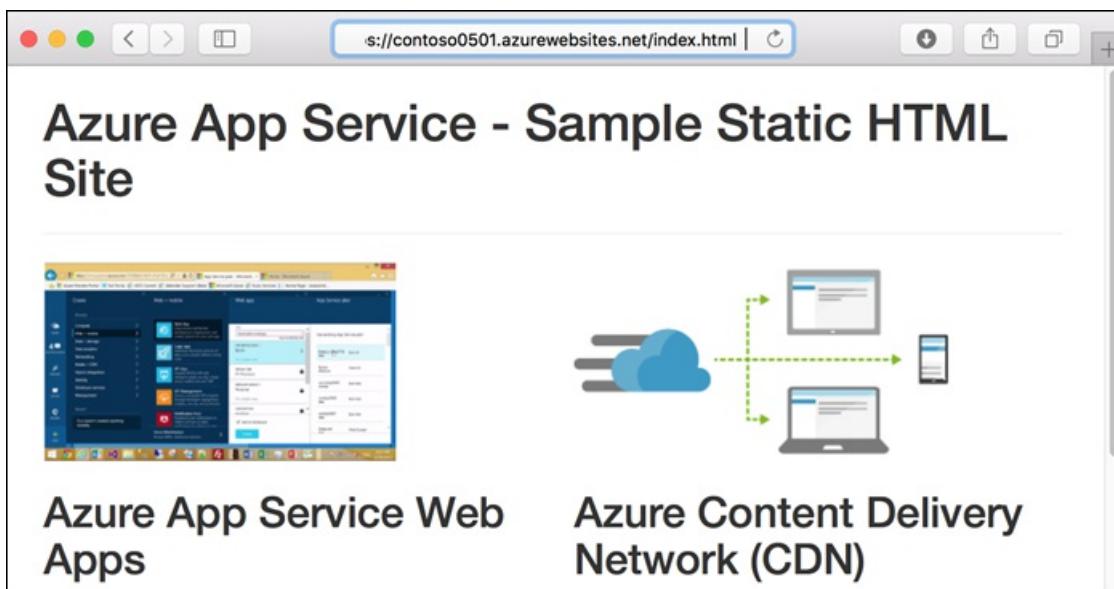
- [Use an SSL certificate in your application code](#)
- [FAQ : App Service Certificates](#)

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](#).

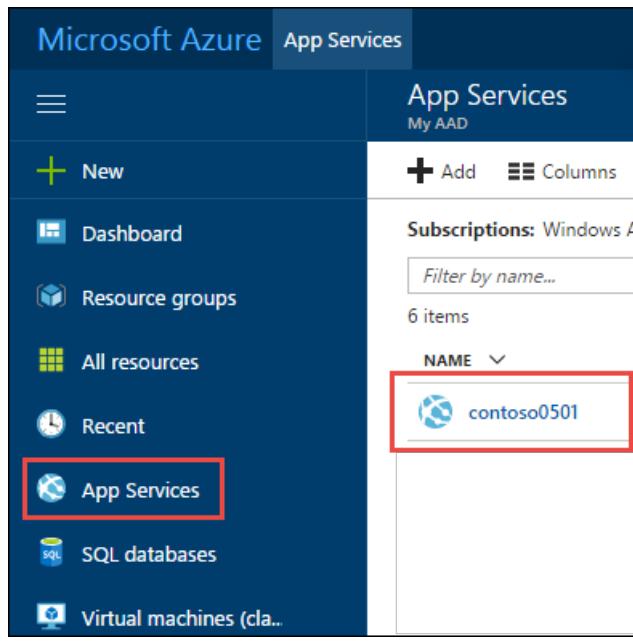
Dynamic site acceleration optimization

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.

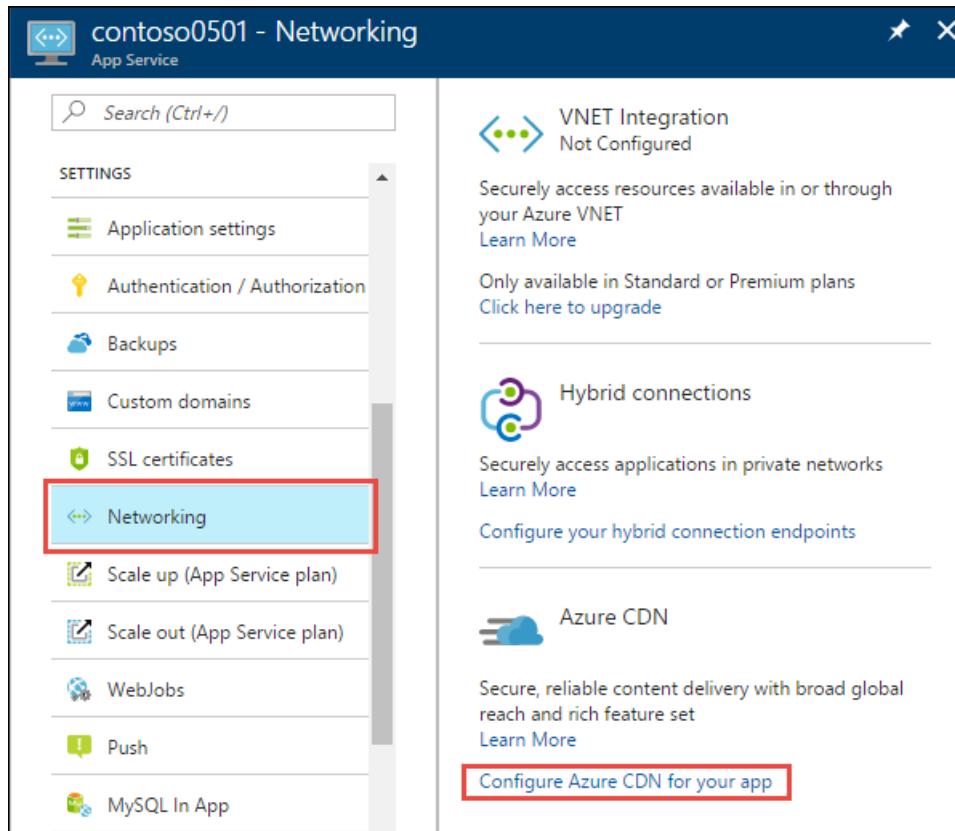
Create a CDN profile and endpoint

In the left navigation, select **App Services**, and then select the app that you created in the [static HTML quickstart](#).



The screenshot shows the Microsoft Azure portal's App Services blade. On the left, there's a sidebar with links like 'New', 'Dashboard', 'Resource groups', 'All resources', 'Recent', and 'App Services'. The 'App Services' link is highlighted with a red box. The main pane lists 'Subscriptions: Windows A...' with a filter input and '6 items'. Below that is a table with a single row for 'contoso0501', which is also highlighted with a red box.

In the **App Service** page, in the **Settings** section, select **Networking > Configure Azure CDN for your app**.



The screenshot shows the 'Networking' settings page for the 'contoso0501' app service. The left sidebar has a 'SETTINGS' section with links like 'Application settings', 'Authentication / Authorization', 'Backups', 'Custom domains', 'SSL certificates', 'Networking' (which is highlighted with a red box), 'Scale up (App Service plan)', 'Scale out (App Service plan)', 'WebJobs', 'Push', and 'MySQL In App'. The main pane shows three sections: 'VNET Integration' (Not Configured), 'Hybrid connections', and 'Azure CDN'. The 'Azure CDN' section contains text about secure content delivery and a 'Configure Azure CDN for your app' button, which is also highlighted with a red box.

In the **Azure Content Delivery Network** page, provide the **New endpoint** settings as specified in the table.

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
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
CDN profile	myCDNProfile	A CDN profile is a collection of CDN endpoints with the same pricing tier.
Pricing tier	Standard Akamai	The pricing tier specifies the provider and available features. This tutorial uses <i>Standard Akamai</i> .
CDN endpoint name	Any name that is unique in the azureedge.net domain	You access your cached resources at the domain <endpointname>.azureedge.net.

Select **Create** to create a CDN profile.

Azure creates the profile and endpoint. The new endpoint appears in the **Endpoints** list, and when it's provisioned, the status is **Running**.

Azure CDN

Azure Content Delivery Network

The Azure Content Delivery Network (CDN) is designed to send audio, video, images, and other files faster and more reliably to customers using servers that are closest to the users. This dramatically increases speed and availability, resulting in significant user experience improvements. [Learn more](#)

Endpoints

HOSTNAME	STATUS	PROTOCOL	...
contoso0501.azureedge.net	Running	HTTP, HTTPS	...

Test the CDN endpoint

Because it takes time for the registration to propagate, the endpoint isn't immediately available for use:

- For **Azure CDN Standard from Microsoft** profiles, propagation usually completes in 10 minutes.
- For **Azure CDN Standard from Akamai** profiles, propagation usually completes within one minute.
- For **Azure CDN Standard from Verizon** and **Azure CDN Premium from Verizon** profiles, propagation usually completes within 90 minutes.

The sample app has an *index.html* file and *css*, *img*, and *js* folders that contain other static assets. The content paths for all of these files are the same at the CDN endpoint. For example, both of the following URLs access the *bootstrap.css* file in the *css* folder:

`http://<appname>.azurewebsites.net/css/bootstrap.css`

`http://<endpointname>.azureedge.net/css/bootstrap.css`

Navigate a browser to the following URL:

`http://<endpointname>.azureedge.net/index.html`

Azure App Service - Sample Static HTML Site

Azure App Service Web Apps

Azure Content Delivery Network (CDN)

You see the same page that you ran earlier in an Azure web app. Azure CDN has retrieved the origin web app's assets and is serving them from the CDN endpoint

To ensure that this page is cached in the CDN, refresh the page. Two requests for the same asset are sometimes required for the CDN to cache the requested content.

For more information about creating Azure CDN profiles and endpoints, see [Getting started with Azure CDN](#).

Purge the CDN

The CDN periodically refreshes its resources from the origin web app based on the time-to-live (TTL) configuration. The default TTL is seven days.

At times you might need to refresh the CDN before the TTL expiration; for example, when you deploy updated content to the web app. To trigger a refresh, manually purge the CDN resources.

In this section of the tutorial, you deploy a change to the web app and purge the CDN to trigger the CDN to refresh its cache.

Deploy a change to the web app

Open the `index.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)

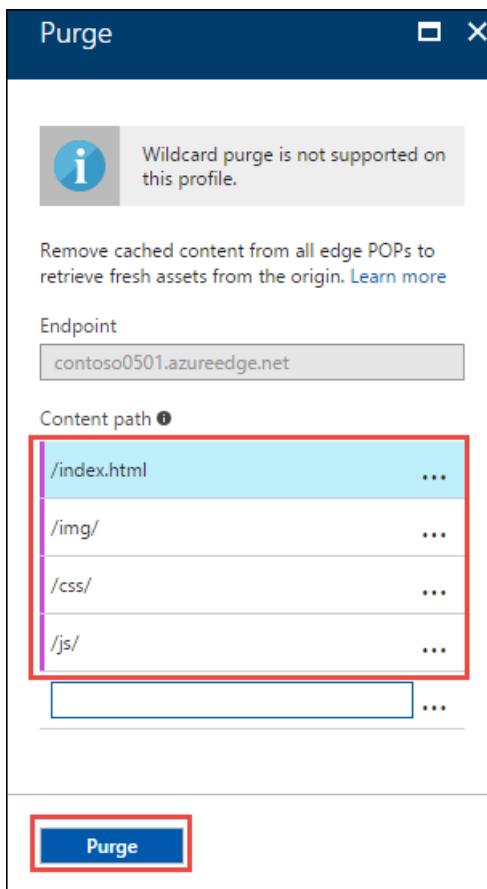
Custom domain Purge Load Stop

Essentials

Resource group myResourceGroup Status Running

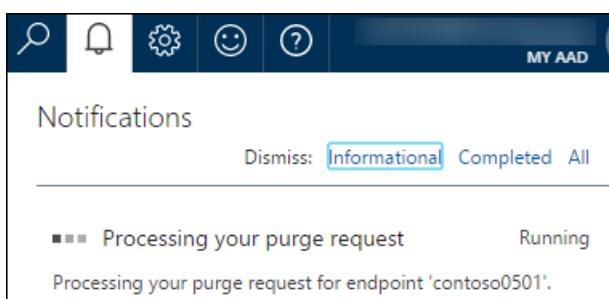
Enter the content paths you want to purge. You can pass a complete file path to purge an individual file, or a path segment to purge and refresh all content in a folder. Because you changed *index.html*, ensure that is in one of the paths.

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:

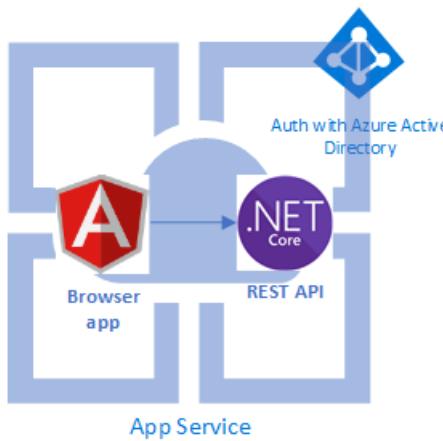
[Tutorial: Add a custom domain to your Azure CDN endpoint](#)

Tutorial: Authenticate and authorize users end-to-end in Azure App Service

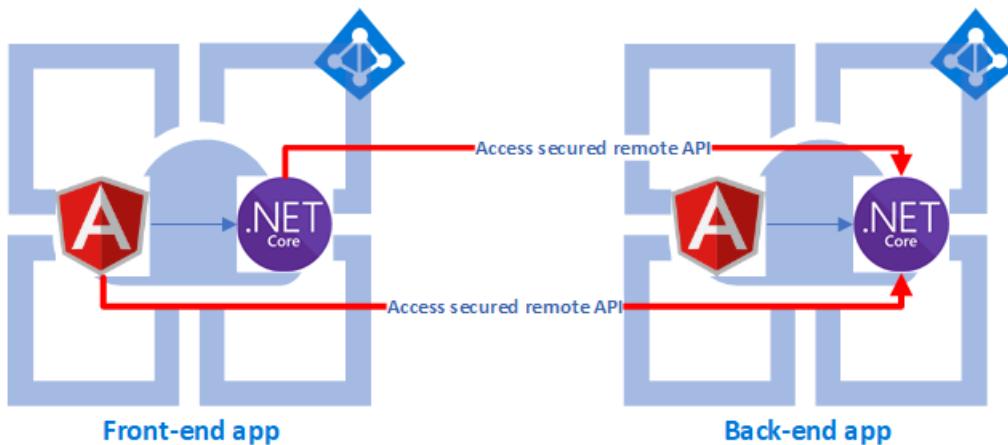
12/31/2019 • 14 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. In addition, App Service has built-in support for [user authentication and authorization](#). This tutorial shows how to secure your apps with App Service authentication and authorization. It uses 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

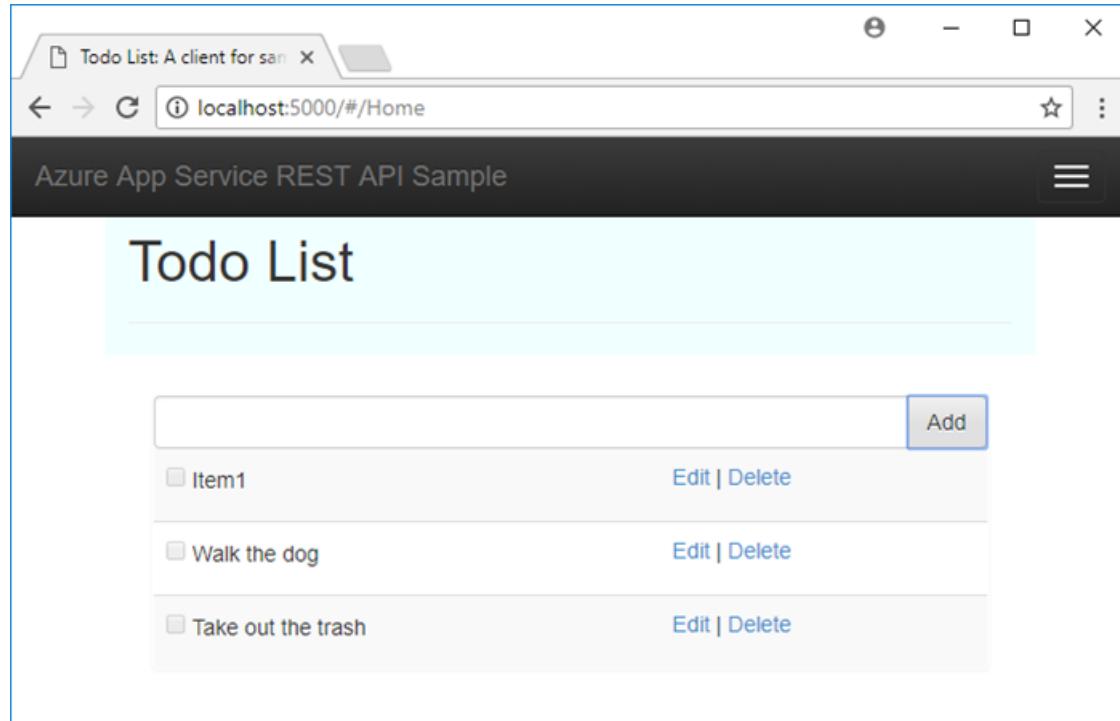
In this step, you set up the local .NET Core project. You use the same project to deploy a back-end API app and a front-end web app.

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.

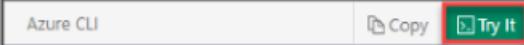


To stop ASP.NET Core at any time, press [Ctrl+C](#) in the terminal.

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

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

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

Deploy apps to Azure

In this step, you deploy the project to two App Service apps. One is the front-end app and the other is the back-end app.

Configure a deployment user

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

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

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

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

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

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

Create Azure resources

In the Cloud Shell, run the following commands to create two 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 [RESTful API with CORS in Azure App Service](#).

```
az group create --name myAuthResourceGroup --location "West Europe"
az appservice plan create --name myAuthAppServicePlan --resource-group myAuthResourceGroup --sku FREE
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <front-end-app-name>
--deployment-local-git --query deploymentLocalGitUrl
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <back-end-app-name> -
--deployment-local-git --query deploymentLocalGitUrl
```

NOTE

Save the URLs of the Git remotes for your front-end and back-end apps, 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
```

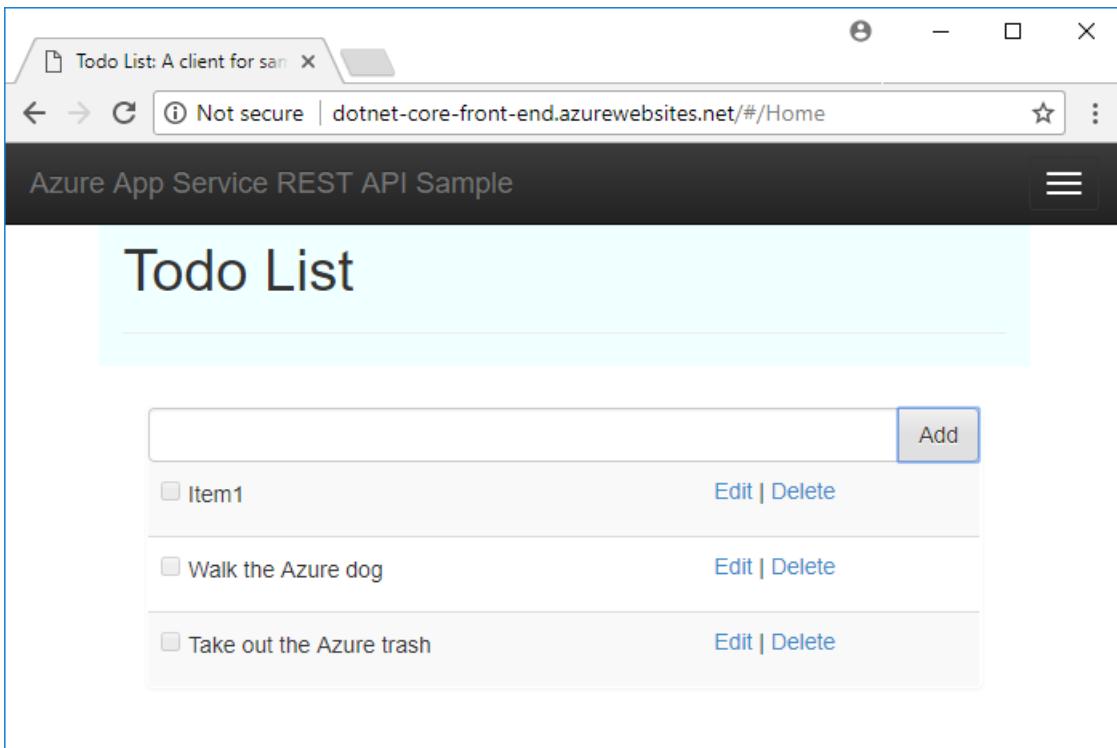
In the local terminal window, run the following Git commands to deploy the same code to the front-end app. Replace *<deploymentLocalGitUrl-of-front-end-app>* with the URL of the Git remote that you saved from [Create Azure resources](#).

```
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.

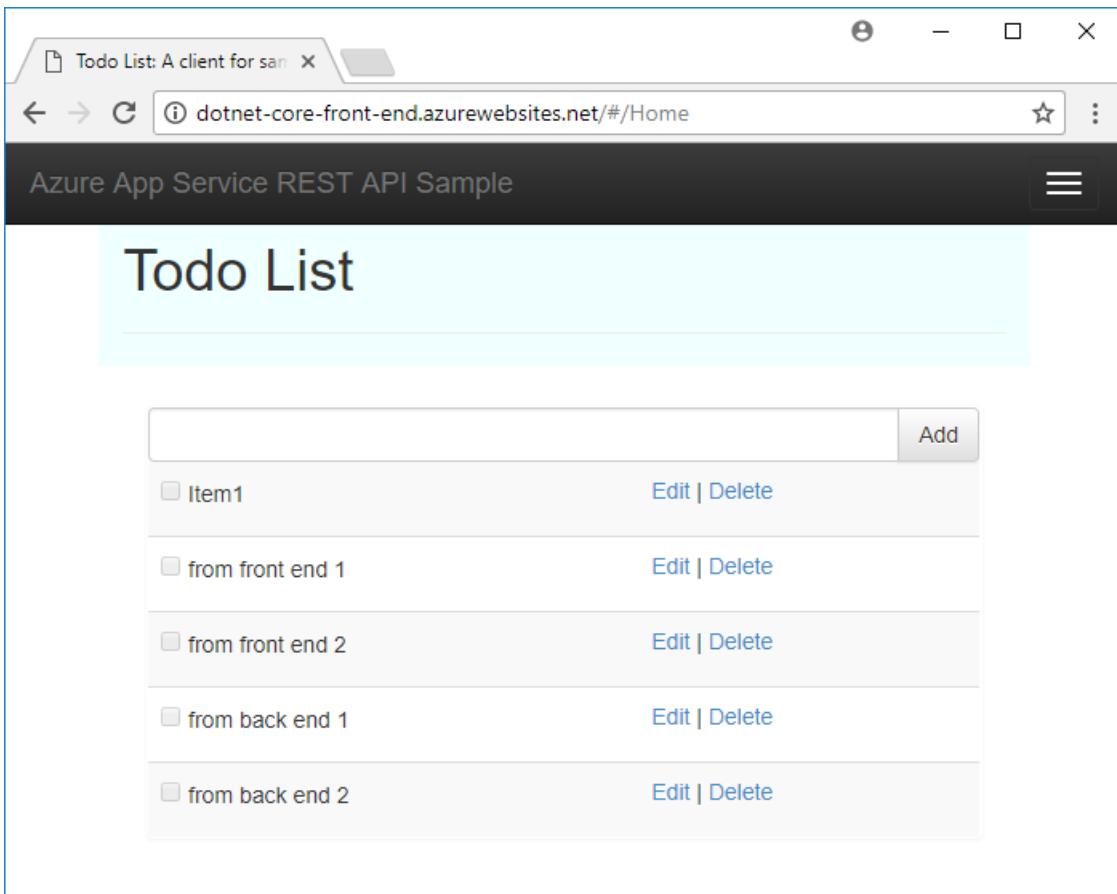
Save all your changes. In the local terminal window, deploy your changes to the front-end app with the following Git commands:

```
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.



Configure auth

In this step, you enable authentication and authorization for the two apps. You also configure the front-end app to generate an access token that you can use to make authenticated calls to the back-end app.

You use Azure Active Directory as the identity provider. For more information, see [Configure Azure Active Directory authentication for your App Services application](#).

Enable authentication and authorization for back-end app

1. In the [Azure portal](#) menu, select **Resource groups** or search for and select *Resource groups* from any page.
2. In **Resource groups**, find and select your resource group. In **Overview**, select your back-end app's management page.

Home > Resource groups > myAuthResourceGroup

myAuthResourceGroup

Resource group

Overview

- Activity log
- Access control (IAM)
- Tags
- Events
- Settings
 - Quickstart
 - Deployments
 - Policies
 - Properties
 - Locks
 - Export template
- Cost Management
 - Cost analysis
 - Cost alerts
 - Budgets
 - Advisor recommendations
- Monitoring
 - Insights (preview)

Subscription (change)
Contoso, Ltd. Subscription
Subscription ID
d6885c49-b217-485c-b854-40e6b3c62c06
Tags (change)
Click here to add tags

Filter by name... Type == all Location == all Add filter No grouping

Showing 1 to 5 of 5 records. □ Show hidden types ⓘ

Name	Type	Location	⋮
dotnet-core-back-end	App Service	Central US	⋮
dotnet-core-front-end	App Service	Central US	⋮
myAuthAppServicePlan	App Service plan	Central US	⋮

3. In your back-end app's left menu, select **Authentication / Authorization**, then enable App Service Authentication by selecting **On**.
4. In **Action to take when request is not authenticated**, select **Log in with Azure Active Directory**.
5. Under **Authentication Providers**, select **Azure Active Directory**

Home > Resource groups > myAuthResourceGroup > dotnet-core-back-end - Authentication / Authorization

dotnet-core-back-end - Authentication / Authorization

App Service

Authentication / Authorization

To enable Authentication / Authorization, please ensure all your custom domains have corresponding SSL bindings, your .NET version is configured to "4.5" or higher and manage pipeline mode is set to "Integrated"

App Service Authentication **On**

Action to take when request is not authenticated **Log in with Azure Active Directory**

Authentication Providers

- Azure Active Directory
Not Configured
- Microsoft
Not Configured
- Facebook

6. Select **Express**, then accept the default settings to create a new AD app and select **OK**.
7. In the **Authentication / Authorization** page, select **Save**.

Once you see the notification with the message

Successfully saved the Auth Settings for <back-end-app-name> App , refresh the page.

8. Select **Azure Active Directory** again, and then select the **Azure AD App**.

9. Copy the **Client ID** of the Azure AD application to a notepad. You need this value later.

The screenshot shows two side-by-side browser windows. The left window is titled 'Azure Active Directory Settings' and contains sections for 'Active Directory Authentication' and 'Express mode'. It shows a list of 'Current Active Directory' applications, with one named 'dotnet-core-back-end' highlighted by a red box. The right window is titled 'Azure AD Applications' and shows a search bar with 'dotnet-core-back-end'. Below it is a table with columns 'APP NAME' and 'CLIENT ID', where the row for 'dotnet-core-back-end' has its 'CLIENT ID' value 'd86ca7ce-55b2-495c-94f2-4b09b25b5164' highlighted by a red box.

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.)

1. In the [Azure portal](#) menu, select **Azure Active Directory** or search for and select *Azure Active Directory* from any page.
2. Select **App registrations > Owned applications**. Select your front-end app name, then select **API permissions**.

3. Select **Add a permission**, then select **My APIs > <back-end-app-name>**.
4. 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,
```

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 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 `resolve` mapping that calls `/auth/me` and sets the access token. It makes sure you have the access token before instantiating the `todoListCtrl` controller. That way all API calls by the controller includes the token.

Deploy updates and test

Save all your changes. In the local terminal window, deploy your changes to the front-end app with the following Git commands:

```
git add .
git commit -m "add authorization header for Angular"
git push frontend 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 web 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.

Create app	
Create an app and deploy files with FTP	Creates an App Service app and deploys a file to it using FTP.
Create an app and deploy code from GitHub	Creates an App Service app and deploys code from a public GitHub repository.
Create an app with continuous deployment from GitHub	Creates an App Service app with continuous publishing from a GitHub repository you own.
Create an app and deploy code from a local Git repository	Creates an App Service app and configures code push from a local Git repository.
Create an app and deploy code to a staging environment	Creates an App Service app with a deployment slot for staging code changes.
Create an ASP.NET Core app in a Docker container	Creates an App Service app on Linux and loads a Docker image from Docker Hub.
Configure app	
Map a custom domain to an app	Creates an App Service app and maps a custom domain name to it.
Bind a custom SSL certificate to an app	Creates an App Service app and binds the SSL certificate of a custom domain name to it.
Scale app	
Scale an app manually	Creates an App Service app and scales it across 2 instances.
Scale an app worldwide with a high-availability architecture	Creates two App Service apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
Protect app	
Integrate with Azure Application Gateway	Creates an App Service app and integrates it with Application Gateway using service endpoint and access restrictions.
Connect app to resources	
Connect an app to a SQL Database	Creates an App Service app and a SQL database, then adds the database connection string to the app settings.

Connect an app to a storage account	Creates an App Service app and a storage account, then adds the storage connection string to the app settings.
Connect an app to an Azure Cache for Redis	Creates an App Service app and an Azure Cache for Redis, then adds the redis connection details to the app settings.)
Connect an app to Cosmos DB	Creates an App Service app and a Cosmos DB, then adds the Cosmos DB connection details to the app settings.
Back up and restore app	
Back up an app	Creates an App Service app and creates a one-time backup for it.
Create a scheduled backup for an app	Creates an App Service app and creates a scheduled backup for it.
Restores an app from a backup	Restores an App Service app from a backup.
Monitor app	
Monitor an app with web server logs	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.

Create app	
Create an app with deployment from GitHub	Creates an App Service app that pulls code from GitHub.
Create an app with continuous deployment from GitHub	Creates an App Service app that continuously deploys code from GitHub.
Create an app and deploy code with FTP	Creates an App Service app and upload files from a local directory using FTP.
Create an app and deploy code from a local Git repository	Creates an App Service app and configures code push from a local Git repository.
Create an app and deploy code to a staging environment	Creates an App Service app with a deployment slot for staging code changes.
Configure app	
Map a custom domain to an app	Creates an App Service app and maps a custom domain name to it.
Bind a custom SSL certificate to an app	Creates an App Service app and binds the SSL certificate of a custom domain name to it.
Scale app	
Scale an app manually	Creates an App Service app and scales it across 2 instances.
Scale an app worldwide with a high-availability architecture	Creates two App Service apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
Connect app to resources	
Connect an app to a SQL Database	Creates an App Service app and a SQL database, then adds the database connection string to the app settings.
Connect an app to a storage account	Creates an App Service app and a storage account, then adds the storage connection string to the app settings.
Back up and restore app	
Back up an app	Creates an App Service app and creates a one-time backup for it.

Create a scheduled backup for an app	Creates an App Service app and creates a scheduled backup for it.
Delete a backup for an app	Deletes an existing backup for an app.
Restore an app from backup	Restores an app from a previously completed backup.
Restore a backup across subscriptions	Restores a web app from a backup in another subscription.
Monitor app	
Monitor an app with web server logs	Creates an App Service app, enables logging for it, and downloads the logs to your local machine.

Azure Resource Manager templates for App Service

12/10/2019 • 2 minutes to read • [Edit Online](#)

The following table includes links to Azure Resource Manager templates for Azure App Service. For recommendations about avoiding common errors when you're creating app templates, see [Guidance on deploying apps with Azure Resource Manager templates](#).

To learn about the JSON syntax and properties for App Services resources, see [Microsoft.Web resource types](#).

Deploying an app	
App Service plan and basic Linux app	Deploys an App Service app that is configured for Linux.
App Service plan and basic Windows app	Deploys an App Service app that is configured for Windows.
App linked to a GitHub repository	Deploys an App Service app that pulls code from GitHub.
App with custom deployment slots	Deploys an App Service app with custom deployment slots/environments.
Configuring an app	
App certificate from Key Vault	Deploys an App Service app certificate from an Azure Key Vault secret and uses it for SSL binding.
App with a custom domain	Deploys an App Service app with a custom host name.
App with a custom domain and SSL	Deploys an App Service app with a custom host name, and gets an app certificate from Key Vault for SSL binding.
App with a GoLang extension	Deploys an App Service app with the Golang site extension. You can then run web applications developed on Golang on Azure.
App with Java 8 and Tomcat 8	Deploys an App Service app with Java 8 and Tomcat 8 enabled. You can then run Java applications in Azure.
Protecting an app	
App integrated with Azure Application Gateway	Deploys an App Service app and an Application Gateway, and isolates the traffic using service endpoint and access restrictions.
Linux app with connected resources	
App on Linux with MySQL	Deploys an App Service app on Linux with Azure Database for MySQL.
App on Linux with PostgreSQL	Deploys an App Service app on Linux with Azure Database for PostgreSQL.

App with connected resources	
App with MySQL	Deploys an App Service app on Windows with Azure Database for MySQL.
App with PostgreSQL	Deploys an App Service app on Windows with Azure Database for PostgreSQL.
App with a SQL database	Deploys an App Service app and a SQL database at the Basic service level.
App with a Blob storage connection	Deploys an App Service app with an Azure Blob storage connection string. You can then use Blob storage from the app.
App with an Azure Cache for Redis	Deploys an App Service app with an Azure Cache for Redis.
App Service Environment	
Create an App Service environment v2	Creates an App Service environment v2 in your virtual network.
Create an App Service environment v2 with an ILB address	Creates an App Service environment v2 in your virtual network with a private internal load balancer address.
Configure the default SSL certificate for an ILB App Service environment or an ILB App Service environment v2	Configures the default SSL certificate for an ILB App Service environment or an ILB App Service environment v2.

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](#)

Operating system functionality on Azure App Service

12/2/2019 • 10 minutes to read • [Edit Online](#)

This article describes the common baseline operating system functionality that is available to all Windows apps running on [Azure App Service](#). This functionality includes file, network, and registry access, and diagnostics logs and events.

NOTE

[Linux apps](#) in App Service run in their own containers. No access to the host operating system is allowed, you do have root access to the container. Likewise, for [apps running in Windows containers](#), you have administrative access to the container but no access to the host operating system.

App Service plan tiers

App Service runs customer apps in a multi-tenant hosting environment. Apps deployed in the **Free** and **Shared** tiers run in worker processes on shared virtual machines, while apps deployed in the **Standard** and **Premium** tiers run on virtual machine(s) dedicated specifically for the apps associated with a single customer.

NOTE

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

Because App Service supports a seamless scaling experience between different tiers, the security configuration enforced for App Service apps remains the same. This ensures that apps don't suddenly behave differently, failing in unexpected ways, when App Service plan switches from one tier to another.

Development frameworks

App Service pricing tiers control the amount of compute resources (CPU, disk storage, memory, and network egress) available to apps. However, the breadth of framework functionality available to apps remains the same regardless of the scaling tiers.

App Service supports a variety of development frameworks, including ASP.NET, classic ASP, node.js, PHP, and Python - all of which run as extensions within IIS. In order to simplify and normalize security configuration, App Service apps typically run the various development frameworks with their default settings. One approach to configuring apps could have been to customize the API surface area and functionality for each individual development framework. App Service instead takes a more generic approach by enabling a common baseline of operating system functionality regardless of an app's development framework.

The following sections summarize the general kinds of operating system functionality available to App Service apps.

File access

Various drives exist within App Service, including local drives and network drives.

Local drives

At its core, App Service is a service running on top of the Azure PaaS (platform as a service) infrastructure. As a result, the local drives that are "attached" to a virtual machine are the same drive types available to any worker role running in Azure. This includes:

- An operating system drive (the D:\ drive)
- An application drive that contains Azure Package cspkg files used exclusively by App Service (and inaccessible to customers)
- A "user" drive (the C:\ drive), whose size varies depending on the size of the VM.

It is important to monitor your disk utilization as your application grows. If the disk quota is reached, it can have adverse effects to your application. For example:

- The app may throw an error indicating not enough space on the disk.
- You may see disk errors when browsing to the Kudu console.
- Deployment from Azure DevOps or Visual Studio may fail with
`ERROR_NOT_ENOUGH_DISK_SPACE: Web deployment task failed. (Web Deploy detected insufficient space on disk)`.
- Your app may suffer slow performance.

Network drives (aka UNC shares)

One of the unique aspects of App Service that makes app deployment and maintenance straightforward is that all user content is stored on a set of UNC shares. This model maps well to the common pattern of content storage used by on-premises web hosting environments that have multiple load-balanced servers.

Within App Service, there is a number of UNC shares created in each data center. A percentage of the user content for all customers in each data center is allocated to each UNC share. Furthermore, all of the file content for a single customer's subscription is always placed on the same UNC share.

Due to how Azure services work, the specific virtual machine responsible for hosting a UNC share will change over time. It is guaranteed that UNC shares will be mounted by different virtual machines as they are brought up and down during the normal course of Azure operations. For this reason, apps should never make hard-coded assumptions that the machine information in a UNC file path will remain stable over time. Instead, they should use the convenient *faux* absolute path **D:\home\site** that App Service provides. This faux absolute path provides a portable, app-and-user-agnostic method for referring to one's own app. By using **D:\home\site**, one can transfer shared files from app to app without having to configure a new absolute path for each transfer.

Types of file access granted to an app

Each customer's subscription has a reserved directory structure on a specific UNC share within a data center. A customer may have multiple apps created within a specific data center, so all of the directories belonging to a single customer subscription are created on the same UNC share. The share may include directories such as those for content, error and diagnostic logs, and earlier versions of the app created by source control. As expected, a customer's app directories are available for read and write access at runtime by the app's application code.

On the local drives attached to the virtual machine that runs an app, App Service reserves a chunk of space on the C:\ drive for app-specific temporary local storage. Although an app has full read/write access to its own temporary local storage, that storage really isn't intended to be used directly by the application code. Rather, the intent is to provide temporary file storage for IIS and web application frameworks. App Service also limits the amount of temporary local storage available to each app to prevent individual apps from consuming excessive amounts of local file storage.

Two examples of how App Service uses temporary local storage are the directory for temporary ASP.NET files and the directory for IIS compressed files. The ASP.NET compilation system uses the "Temporary ASP.NET Files" directory as a temporary compilation cache location. IIS uses the "IIS Temporary Compressed Files" directory to store compressed response output. Both of these types of file usage (as well as others) are remapped in App Service to per-app temporary local storage. This remapping ensures that functionality continues as expected.

Each app in App Service runs as a random unique low-privileged worker process identity called the "application pool identity", described further here: <https://www.iis.net/learn/manage/configuring-security/application-pool-identities>. Application code uses this identity for basic read-only access to the operating system drive (the D:\ drive). This means application code can list common directory structures and read common files on operating system drive. Although this might appear to be a somewhat broad level of access, the same directories and files are accessible when you provision a worker role in an Azure hosted service and read the drive contents.

File access across multiple instances

The home directory contains an app's content, and application code can write to it. If an app runs on multiple instances, the home directory is shared among all instances so that all instances see the same directory. So, for example, if an app saves uploaded files to the home directory, those files are immediately available to all instances.

Network access

Application code can use TCP/IP and UDP-based protocols to make outbound network connections to Internet accessible endpoints that expose external services. Apps can use these same protocols to connect to services within Azure, for example, by establishing HTTPS connections to SQL Database.

There is also a limited capability for apps to establish one local loopback connection, and have an app listen on that local loopback socket. This feature exists primarily to enable apps that listen on local loopback sockets as part of their functionality. Each app sees a "private" loopback connection. App "A" cannot listen to a local loopback socket established by app "B".

Named pipes are also supported as an inter-process communication (IPC) mechanism between different processes that collectively run an app. For example, the IIS FastCGI module relies on named pipes to coordinate the individual processes that run PHP pages.

Code execution, processes, and memory

As noted earlier, apps run inside of low-privileged worker processes using a random application pool identity. Application code has access to the memory space associated with the worker process, as well as any child processes that may be spawned by CGI processes or other applications. However, one app cannot access the memory or data of another app even if it is on the same virtual machine.

Apps can run scripts or pages written with supported web development frameworks. App Service doesn't configure any web framework settings to more restricted modes. For example, ASP.NET apps running on App Service run in "full" trust as opposed to a more restricted trust mode. Web frameworks, including both classic ASP and ASP.NET, can call in-process COM components (but not out of process COM components) like ADO (ActiveX Data Objects) that are registered by default on the Windows operating system.

Apps can spawn and run arbitrary code. It is allowable for an app to do things like spawn a command shell or run a PowerShell script. However, even though arbitrary code and processes can be spawned from an app, executable programs and scripts are still restricted to the privileges granted to the parent application pool. For example, an app can spawn an executable that makes an outbound HTTP call, but that same executable cannot attempt to unbind the IP address of a virtual machine from its NIC. Making an outbound network call is allowed to low-privileged code, but attempting to reconfigure network settings on a virtual machine requires administrative privileges.

Diagnostics logs and events

Log information is another set of data that some apps attempt to access. The types of log information available to code running in App Service includes diagnostic and log information generated by an app that is also easily accessible to the app.

For example, W3C HTTP logs generated by an active app are available either on a log directory in the network

share location created for the app, or available in blob storage if a customer has set up W3C logging to storage. The latter option enables large quantities of logs to be gathered without the risk of exceeding the file storage limits associated with a network share.

In a similar vein, real-time diagnostics information from .NET apps can also be logged using the .NET tracing and diagnostics infrastructure, with options to write the trace information to either the app's network share, or alternatively to a blob storage location.

Areas of diagnostics logging and tracing that aren't available to apps are Windows ETW events and common Windows event logs (for example, System, Application, and Security event logs). Since ETW trace information can potentially be viewable machine-wide (with the right ACLs), read and write access to ETW events are blocked. Developers might notice that API calls to read and write ETW events and common Windows event logs appear to work, but that is because App Service is "faking" the calls so that they appear to succeed. In reality, the application code has no access to this event data.

Registry access

Apps have read-only access to much (though not all) of the registry of the virtual machine they are running on. In practice, this means registry keys that allow read-only access to the local Users group are accessible by apps. One area of the registry that is currently not supported for either read or write access is the HKEY_CURRENT_USER hive.

Write-access to the registry is blocked, including access to any per-user registry keys. From the app's perspective, write access to the registry should never be relied upon in the Azure environment since apps can (and do) get migrated across different virtual machines. The only persistent writeable storage that can be depended on by an app is the per-app content directory structure stored on the App Service UNC shares.

Remote desktop access

App Service doesn't provide remote desktop access to the VM instances.

More information

[Azure App Service sandbox](#) - The most up-to-date information about the execution environment of App Service. This page is maintained directly by the App Service development team.

Deployment Best Practices

1/8/2020 • 4 minutes to read • [Edit Online](#)

Every development team has unique requirements that can make implementing an efficient deployment pipeline difficult on any cloud service. This article introduces the three main components of deploying to App Service: deployment sources, build pipelines, and deployment mechanisms. This article also covers some best practices and tips for specific language stacks.

Deployment Components

Deployment Source

A deployment source is the location of your application code. For production apps, the deployment source is usually a repository hosted by version control software such as [GitHub](#), [BitBucket](#), or [Azure Repos](#). For development and test scenarios, the deployment source may be a [project on your local machine](#). App Service also supports [OneDrive](#) and [Dropbox folders](#) as deployment sources. While cloud folders can make it easy to get started with App Service, it is not typically recommended to use this source for enterprise-level production applications.

Build Pipeline

Once you decide on a deployment source, your next step is to choose a build pipeline. A build pipeline reads your source code from the deployment source and executes a series of steps (such as compiling code, minifying HTML and JavaScript, running tests, and packaging components) to get the application in a runnable state. The specific commands executed by the build pipeline depend on your language stack. These operations can be executed on a build server such as Azure Pipelines, or executed locally.

Deployment Mechanism

The deployment mechanism is the action used to put your built application into the `/home/site/wwwroot` directory of your web app. The `/wwwroot` directory is a mounted storage location shared by all instances of your web app. When the deployment mechanism puts your application in this directory, your instances receive a notification to sync the new files. App Service supports the following deployment mechanisms:

- Kudu endpoints: [Kudu](#) is the open-source developer productivity tool that runs as a separate process in Windows App Service, and as a second container in Linux App Service. Kudu handles continuous deployments and provides HTTP endpoints for deployment, such as `zipdeploy`.
- FTP and WebDeploy: Using your [site or user credentials](#), you can upload files via [FTP](#) or [WebDeploy](#). These mechanisms do not go through Kudu.

Deployment tools such as Azure Pipelines, Jenkins, and editor plugins use one of these deployment mechanisms.

Language-Specific Considerations

Java

Use the Kudu [zipdeploy](#)/ API for deploying JAR applications, and [wardeploy](#)/ for WAR apps. If you are using Jenkins, you can use those APIs directly in your deployment phase. For more information, see [this article](#).

Node

By default, Kudu executes the build steps for your Node application (`npm install`). If you are using a build service such as Azure DevOps, then the Kudu build is unnecessary. To disable the Kudu build, create an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, with a value of `false`.

.NET

By default, Kudu executes the build steps for your .Net application (`dotnet build`). If you are using a build service such as Azure DevOps, then the Kudu build is unnecessary. To disable the Kudu build, create an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, with a value of `false`.

Other Deployment Considerations

Use deployment slots

Whenever possible, use [deployment slots](#) when deploying a new production build. When using a Standard App Service Plan tier or better, you can deploy your app to a staging environment, validate your changes, and do smoke tests. When you are ready, you can swap your staging and production slots. The swap operation warms up the necessary worker instances to match your production scale, thus eliminating downtime.

Local Cache

Azure App Service content is stored on Azure Storage and is surfaced up in a durable manner as a content share. However, some apps just need a high-performance, read-only content store that they can run with high availability. These apps can benefit from using [local cache](#). Local cache is not recommended for content management sites such as WordPress.

Always use local cache in conjunction with [deployment slots](#) to prevent downtime. See [this section](#) for information on using these features together.

High CPU or Memory

If your App Service Plan is using over 90% of available CPU or memory, the underlying virtual machine may have trouble processing your deployment. When this happens, temporarily scale up your instance count to perform the deployment. Once the deployment has finished, you can return the instance count to its previous value.

For more information on best practices, visit [App Service Diagnostics](#) to find out actionable best practices specific to your resource.

- Navigate to your Web App in the [Azure portal](#).
- Click on **Diagnose and solve problems** in the left navigation, which opens App Service Diagnostics.
- Choose **Best Practices** homepage tile.
- Click **Best Practices for Availability & Performance** or **Best Practices for Optimal Configuration** to view the current state of your app in regards to these best practices.

You can also use this link to directly open App Service Diagnostics for your resource:

```
https://ms.portal.azure.com/?websitesextension_ext=asd.featurePath%3Ddetectors%2FparentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/prc
```

Security recommendations for App Service

12/2/2019 • 3 minutes to read • [Edit Online](#)

This article contains security recommendations for Azure App Service. Implementing these recommendations will help you fulfill your security obligations as described in our shared responsibility model and will improve the overall security for your Web App solutions. For more information on what Microsoft does to fulfill service provider responsibilities, read [Azure infrastructure security](#).

General

RECOMMENDATION	COMMENTS
Stay up-to-date	Use the latest versions of supported platforms, programming languages, protocols, and frameworks.

Identity and access management

RECOMMENDATION	COMMENTS
Disable anonymous access	Unless you need to support anonymous requests, disable anonymous access. For more information on Azure App Service authentication options, see Authentication and authorization in Azure App Service .
Require authentication	Whenever possible, use the App Service authentication module instead of writing code to handle authentication and authorization. See Authentication and authorization in Azure App Service .
Protect back-end resources with authenticated access	You can either use the user's identity or use an application identity to authenticate to a back-end resource. When you choose to use an application identity use a managed identity .
Require client certificate authentication	Client certificate authentication improves security by only allowing connections from clients that can authenticate using certificates that you provide.

Data protection

RECOMMENDATION	COMMENTS
Redirect HTTP to HTTPS	By default, clients can connect to web apps by using both HTTP or HTTPS. We recommend redirecting HTTP to HTTPS because HTTPS uses the SSL/TLS protocol to provide a secure connection, which is both encrypted and authenticated.
Encrypt communication to Azure resources	When your app connects to Azure resources, such as SQL Database or Azure Storage , the connection stays in Azure. Since the connection goes through the shared networking in Azure, you should always encrypt all communication.

RECOMMENDATION	COMMENTS
Require the latest TLS version possible	Since 2018 new Azure App Service apps use TLS 1.2. Newer versions of TLS include security improvements over older protocol versions.
Use FTPS	App Service supports both FTP and FTPS for deploying your files. Use FTPS instead of FTP when possible. When one or both of these protocols are not in use, you should disable them .
Secure application data	Don't store application secrets, such as database credentials, API tokens, or private keys in your code or configuration files. The commonly accepted approach is to access them as environment variables using the standard pattern in your language of choice. In Azure App Service, you can define environment variables through app settings and connection strings . App settings and connection strings are stored encrypted in Azure. The app settings are decrypted only before being injected into your app's process memory when the app starts. The encryption keys are rotated regularly. Alternatively, you can integrate your Azure App Service app with Azure Key Vault for advanced secrets management. By accessing the Key Vault with a managed identity , your App Service app can securely access the secrets you need.

Networking

RECOMMENDATION	COMMENTS
Use static IP restrictions	Azure App Service on Windows lets you define a list of IP addresses that are allowed to access your app. The allowed list can include individual IP addresses or a range of IP addresses defined by a subnet mask. For more information, see Azure App Service Static IP Restrictions .
Use the isolated pricing tier	Except for the isolated pricing tier, all tiers run your apps on the shared network infrastructure in Azure App Service. The isolated tier gives you complete network isolation by running your apps inside a dedicated App Service environment . An App Service environment runs in your own instance of Azure Virtual Network .
Use secure connections when accessing on-premises resources	You can use Hybrid connections , Virtual Network integration , or App Service environment's to connect to on-premises resources.
Limit exposure to inbound network traffic	Network security groups allow you to restrict network access and control the number of exposed endpoints. For more information, see How To Control Inbound Traffic to an App Service Environment .

Monitoring

RECOMMENDATION	COMMENTS

RECOMMENDATION	COMMENTS
Use Azure Security Center standard tier	Azure Security Center is natively integrated with Azure App Service. It can run assessments and provide security recommendations.

Next steps

Check with your application provider to see if there are additional security requirements. For more information on developing secure applications, see [Secure Development Documentation](#).

Authentication and authorization in Azure App Service

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

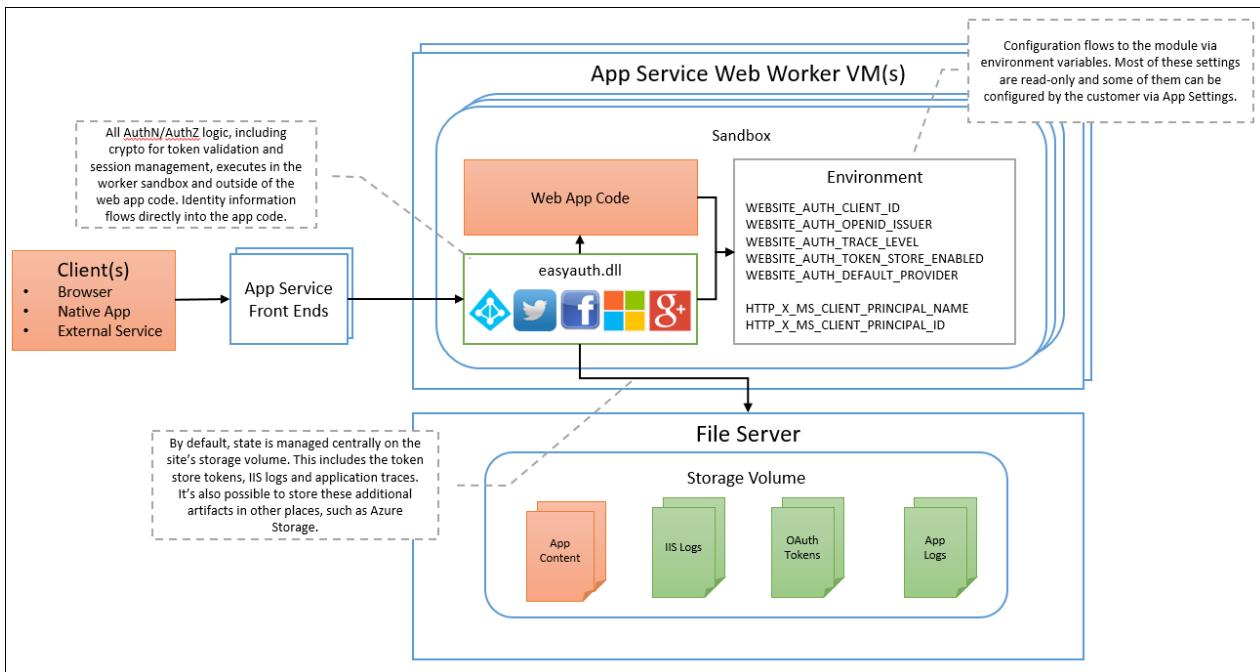
You're not required to use App Service for AuthN/AuthO. You can use the bundled security features in your web framework of choice, or you can write your own utilities. However, keep in mind that [Chrome 80 is making breaking changes to its implementation of SameSite for cookies](#) (release date around 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](#).

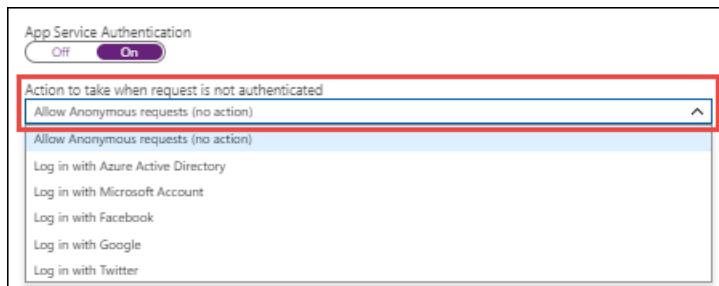
The table below shows the steps of the authentication flow.

STEP	WITHOUT PROVIDER SDK	WITH PROVIDER SDK
1. Sign user in	Redirects client to <code>/auth/login/<provider></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/<provider>/callback</code> .	Client code posts token from provider to <code>/auth/login/<provider></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](#)

OS and runtime patching in Azure App Service

1/28/2020 • 4 minutes to read • [Edit Online](#)

This article shows you how to get certain version information regarding the OS or software in [App Service](#).

App Service is a Platform-as-a-Service, which means that the OS and application stack are managed for you by Azure; you only manage your application and its data. More control over the OS and application stack is available you in [Azure Virtual Machines](#). With that in mind, it is nevertheless helpful for you as an App Service user to know more information, such as:

- How and when are OS updates applied?
- How is App Service patched against significant vulnerabilities (such as zero-day)?
- Which OS and runtime versions are running your apps?

For security reasons, certain specifics of security information are not published. However, the article aims to alleviate concerns by maximizing transparency on the process, and how you can stay up-to-date on security-related announcements or runtime updates.

How and when are OS updates applied?

Azure manages OS patching on two levels, the physical servers and the guest virtual machines (VMs) that run the App Service resources. Both are updated monthly, which aligns to the monthly [Patch Tuesday](#) schedule. These updates are applied automatically, in a way that guarantees the high-availability SLA of Azure services.

For detailed information on how updates are applied, see [Demystifying the magic behind App Service OS updates](#).

How does Azure deal with significant vulnerabilities?

When severe vulnerabilities require immediate patching, such as [zero-day vulnerabilities](#), the high-priority updates are handled on a case-by-case basis.

Stay current with critical security announcements in Azure by visiting [Azure Security Blog](#).

When are supported language runtimes updated, added, or deprecated?

New stable versions of supported language runtimes (major, minor, or patch) are periodically added to App Service instances. Some updates overwrite the existing installation, while others are installed side by side with existing versions. An overwrite installation means that your app automatically runs on the updated runtime. A side-by-side installation means you must manually migrate your app to take advantage of a new runtime version. For more information, see one of the subsections.

Runtime updates and deprecations are announced here:

- <https://azure.microsoft.com/updates/?product=app-service>
- <https://github.com/Azure/app-service-announcements/issues>

NOTE

Information here applies to language runtimes that are built into an App Service app. A custom runtime you upload to App Service, for example, remains unchanged unless you manually upgrade it.

New patch updates

Patch updates to .NET, PHP, Java SDK, or Tomcat/Jetty version are applied automatically by overwriting the existing installation with the new version. Node.js patch updates are installed side by side with the existing versions (similar to major and minor versions in the next section). New Python patch versions can be installed manually through [site extensions](#), side by side with the built-in Python installations.

New major and minor versions

When a new major or minor version is added, it is installed side by side with the existing versions. You can manually upgrade your app to the new version. If you configured the runtime version in a configuration file (such as `web.config` and `package.json`), you need to upgrade with the same method. If you used an App Service setting to configure your runtime version, you can change it in the [Azure portal](#) or by running an [Azure CLI](#) command in the [Cloud Shell](#), as shown in the following examples:

```
az webapp config set --net-framework-version v4.7 --resource-group <groupname> --name <appname>
az webapp config set --php-version 7.0 --resource-group <groupname> --name <appname>
az webapp config appsettings set --settings WEBSITE_NODE_DEFAULT_VERSION=8.9.3 --resource-group <groupname> --
name <appname>
az webapp config set --python-version 3.4 --resource-group <groupname> --name <appname>
az webapp config set --java-version 1.8 --java-container Tomcat --java-container-version 9.0 --resource-group
<groupname> --name <appname>
```

Deprecated versions

When an older version is deprecated, the removal date is announced so that you can plan your runtime version upgrade accordingly.

How can I query OS and runtime update status on my instances?

While critical OS information is locked down from access (see [Operating system functionality on Azure App Service](#)), the [Kudu console](#) enables you to query your App Service instance regarding the OS version and runtime versions.

The following table shows how to the versions of Windows and of the language runtime that are running your apps:

INFORMATION	WHERE TO FIND IT
Windows version	See <a href="https://<appname>.scm.azurewebsites.net/Env.cshtml">https://<appname>.scm.azurewebsites.net/Env.cshtml (under System info)
.NET version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>powershell -command "gci 'Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Net Framework Setup\NDP\CDF'"</code>
.NET Core version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>dotnet --version</code>
PHP version	At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt: <code>php --version</code>

INFORMATION	WHERE TO FIND IT
Default Node.js version	<p>In the Cloud Shell, run the following command:</p> <pre data-bbox="822 208 1393 280">az webapp config appsettings list --resource-group <groupname> --name <appname> --query "[? name=='WEBSITE_NODE_DEFAULT_VERSION']"</pre>
Python version	<p>At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt:</p> <pre data-bbox="822 433 1012 460">python --version</pre>
Java version	<p>At <a href="https://<appname>.scm.azurewebsites.net/DebugConsole">https://<appname>.scm.azurewebsites.net/DebugConsole , run the following command in the command prompt:</p> <pre data-bbox="822 613 980 640">java -version</pre>

NOTE

Access to registry location

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\Packages` , where information on "KB" patches is stored, is locked down.

More resources

[Trust Center: Security](#)

[64 bit ASP.NET Core on Azure App Service](#)

Security controls for Azure App Service

12/2/2019 • 3 minutes to read • [Edit Online](#)

This article documents the security controls built into Azure App Service.

A security control is a quality or feature of an Azure service that contributes to the service's ability to prevent, detect, and respond to security vulnerabilities.

For each control, we use "Yes" or "No" to indicate whether it is currently in place for the service, "N/A" for a control that is not applicable to the service. We might also provide a note or links to more information about an attribute.

Network

SECURITY CONTROL	YES/NO	NOTES	DOCUMENTATION
Service endpoint support	Yes	Available for App Service.	Azure App Service Access Restrictions
VNet injection support	Yes	App Service Environments are private implementations of App Service dedicated to a single customer injected into a customer's virtual network.	Introduction to the App Service Environments
Network Isolation and Firewalling support	Yes	For the public multi-tenant variation of App Service, customers can configure network ACLs (IP Restrictions) to lock down allowed inbound traffic. App Service Environments are deployed directly into virtual networks and hence can be secured with NSGs.	Azure App Service Access Restrictions
Forced tunneling support	Yes	App Service Environments can be deployed into a customer's virtual network where forced tunneling is configured.	Configure your App Service Environment with forced tunneling

Monitoring & logging

SECURITY CONTROL	YES/NO	NOTES	DOCUMENTATION
------------------	--------	-------	---------------

SECURITY CONTROL	YES/NO	NOTES	DOCUMENTATION
Azure monitoring support (Log analytics, App insights, etc.)	Yes	App Service integrates with Application Insights for languages that support Application Insights (Full .NET Framework, .NET Core, Java and Node.js). See Monitor Azure App Service performance . App Service also sends application metrics into Azure Monitor.	Monitor apps in Azure App Service
Control and management plane logging and audit	Yes	All management operations performed on App Service objects occur via Azure Resource Manager . Historical logs of these operations are available both in the portal and via the CLI.	Azure Resource Manager resource provider operations, az monitor activity-log
Data plane logging and audit	No	The data plane for App Service is a remote file share containing a customer's deployed web site content. There is no auditing of the remote file share.	

Identity

SECURITY CONTROL	YES/NO	NOTES	DOCUMENTATION
Authentication	Yes	Customers can build applications on App Service that automatically integrate with Azure Active Directory (Azure AD) as well as other OAuth compatible identity providers. For management access to App Service assets, all access is controlled by a combination of Azure AD authenticated principal and Azure Resource Manager RBAC roles.	Authentication and authorization in Azure App Service
Authorization	Yes	For management access to App Service assets, all access is controlled by a combination of Azure AD authenticated principal and Azure Resource Manager RBAC roles.	Authentication and authorization in Azure App Service

Data protection

SECURITY CONTROL	YES/NO	NOTES	DOCUMENTATION
Server-side encryption at rest: Microsoft-managed keys	Yes	<p>Web site file content is stored in Azure Storage, which automatically encrypts the content at rest.</p> <p>Customer supplied secrets are encrypted at rest. The secrets are encrypted at rest while stored in App Service configuration databases.</p> <p>Locally attached disks can optionally be used as temporary storage by websites (D:\local and %TMP%). Locally attached disks are not encrypted at rest.</p>	Azure Storage encryption for data at rest
Server-side encryption at rest: customer-managed keys (BYOK)	Yes	Customers can choose to store application secrets in Key Vault and retrieve them at runtime.	Use Key Vault references for App Service and Azure Functions (preview)
Column level encryption (Azure Data Services)	N/A		
Encryption in transit (such as ExpressRoute encryption, in VNet encryption, and VNet-VNet encryption)	Yes	Customers can configure web sites to require and use HTTPS for inbound traffic.	How to make an Azure App Service HTTPS only (blog post)
API calls encrypted	Yes	Management calls to configure App Service occur via Azure Resource Manager calls over HTTPS.	

Configuration management

SECURITY CONTROL	YES/NO	NOTES	DOCUMENTATION
Configuration management support (versioning of configuration, etc.)	Yes	For management operations, the state of an App Service configuration can be exported as an Azure Resource Manager template and versioned over time. For runtime operations, customers can maintain multiple different live versions of an application using the App Service deployment slots feature.	

Next steps

- Learn more about the [built-in security controls across Azure services](#).

Security in Azure App Service

12/2/2019 • 7 minutes to read • [Edit Online](#)

This article shows you how [Azure App Service](#) helps secure your web app, mobile app back end, API app, and [function app](#). It also shows how you can further secure your app with the built-in App Service features.

The platform components of App Service, including Azure VMs, storage, network connections, web frameworks, management and integration features, are actively secured and hardened. App Service goes through vigorous compliance checks on a continuous basis to make sure that:

- Your app resources are [secured](#) from the other customers' Azure resources.
- [VM instances and runtime software are regularly updated](#) to address newly discovered vulnerabilities.
- Communication of secrets (such as connection strings) between your app and other Azure resources (such as [SQL Database](#)) stays within Azure and doesn't cross any network boundaries. Secrets are always encrypted when stored.
- All communication over the App Service connectivity features, such as [hybrid connection](#), is encrypted.
- Connections with remote management tools like Azure PowerShell, Azure CLI, Azure SDKs, REST APIs, are all encrypted.
- 24-hour threat management protects the infrastructure and platform against malware, distributed denial-of-service (DDoS), man-in-the-middle (MITM), and other threats.

For more information on infrastructure and platform security in Azure, see [Azure Trust Center](#).

The following sections show you how to further protect your App Service app from threats.

HTTPS and Certificates

App Service lets you secure your apps with [HTTPS](#). When your app is created, its default domain name (<app_name>.azurewebsites.net) is already accessible using HTTPS. If you [configure a custom domain for your app](#), you should also [secure it with an SSL certificate](#) so that client browsers can make secured HTTPS connections to your custom domain. There are several types of certificates supported by App Service:

- Free App Service Managed Certificate
- App Service certificate
- Third-party certificate
- Certificate imported from Azure Key Vault

For more information, see [Add an SSL certificate in Azure App Service](#).

Insecure protocols (HTTP, TLS 1.0, FTP)

To secure your app against all unencrypted (HTTP) connections, App Service provides one-click configuration to enforce HTTPS. Unsecured requests are turned away before they even reach your application code. For more information, see [Enforce HTTPS](#).

[TLS](#) 1.0 is no longer considered secure by industry standards, such as [PCI DSS](#). App Service lets you disable outdated protocols by [enforcing TLS 1.1/1.2](#).

App Service supports both FTP and FTPS for deploying your files. However, FTPS should be used instead of FTP, if at all possible. When one or both of these protocols are not in use, you should [disable them](#).

Static IP restrictions

By default, your App Service app accepts requests from all IP addresses from the internet, but you can limit that access to a small subset of IP addresses. App Service on Windows lets you define a list of IP addresses that are allowed to access your app. The allowed list can include individual IP addresses or a range of IP addresses defined by a subnet mask. For more information, see [Azure App Service Static IP Restrictions](#).

For App Service on Windows, you can also restrict IP addresses dynamically by configuring the *web.config*. For more information, see [Dynamic IP Security <dynamicIpSecurity>](#).

Client authentication and authorization

Azure App Service provides turn-key authentication and authorization of users or client apps. When enabled, it can sign in users and client apps with little or no application code. You may implement your own authentication and authorization solution, or allow App Service to handle it for you instead. The authentication and authorization module handles web requests before handing them off to your application code, and it denies unauthorized requests before they reach your code.

App Service authentication and authorization support multiple authentication providers, including Azure Active Directory, Microsoft accounts, Facebook, Google, and Twitter. For more information, see [Authentication and authorization in Azure App Service](#).

Service-to-service authentication

When authenticating against a back-end service, App Service provides two different mechanisms depending on your need:

- **Service identity** - Sign in to the remote resource using the identity of the app itself. App Service lets you easily create a [managed identity](#), which you can use to authenticate with other services, such as [Azure SQL Database](#) or [Azure Key Vault](#). For an end-to-end tutorial of this approach, see [Secure Azure SQL Database connection from App Service using a managed identity](#).
- **On-behalf-of (OBO)** - Make delegated access to remote resources on behalf of the user. With Azure Active Directory as the authentication provider, your App Service app can perform delegated sign-in to a remote service, such as [Azure Active Directory Graph API](#) or a remote API app in App Service. For an end-to-end tutorial of this approach, see [Authenticate and authorize users end-to-end in Azure App Service](#).

Connectivity to remote resources

There are three types of remote resources your app may need to access:

- [Azure resources](#)
- [Resources inside an Azure Virtual Network](#)
- [On-premises resources](#)

In each of these cases, App Service provides a way for you to make secure connections, but you should still observe security best practices. For example, always use encrypted connections even if the back-end resource allows unencrypted connections. Furthermore, make sure that your back-end Azure service allows the minimum set of IP addresses. You can find the outbound IP addresses for your app at [Inbound and outbound IP addresses in Azure App Service](#).

Azure resources

When your app connects to Azure resources, such as [SQL Database](#) and [Azure Storage](#), the connection stays within Azure and doesn't cross any network boundaries. However, the connection goes through the shared networking in Azure, so always make sure that your connection is encrypted.

If your app is hosted in an [App Service environment](#), you should [connect to supported Azure services using Virtual Network service endpoints](#).

Resources inside an Azure Virtual Network

Your app can access resources in an [Azure Virtual Network](#) through [Virtual Network integration](#). The integration is established with a Virtual Network using a point-to-site VPN. The app can then access the resources in the Virtual Network using their private IP addresses. The point-to-site connection, however, still traverses the shared networks in Azure.

To isolate your resource connectivity completely from the shared networks in Azure, create your app in an [App Service environment](#). Since an App Service environment is always deployed to a dedicated Virtual Network, connectivity between your app and resources within the Virtual Network is fully isolated. For other aspects of network security in an App Service environment, see [Network isolation](#).

On-premises resources

You can securely access on-premises resources, such as databases, in three ways:

- [Hybrid connections](#) - Establishes a point-to-point connection to your remote resource through a TCP tunnel. The TCP tunnel is established using TLS 1.2 with shared access signature (SAS) keys.
- [Virtual Network integration](#) with site-to-site VPN - As described in [Resources inside an Azure Virtual Network](#), but the Virtual Network can be connected to your on-premises network through a [site-to-site VPN](#). In this network topology, your app can connect to on-premises resources like other resources in the Virtual Network.
- [App Service environment](#) with site-to-site VPN - As described in [Resources inside an Azure Virtual Network](#), but the Virtual Network can be connected to your on-premises network through a [site-to-site VPN](#). In this network topology, your app can connect to on-premises resources like other resources in the Virtual Network.

Application secrets

Don't store application secrets, such as database credentials, API tokens, and private keys in your code or configuration files. The commonly accepted approach is to access them as [environment variables](#) using the standard pattern in your language of choice. In App Service, the way to define environment variables is through [app settings](#) (and, especially for .NET applications, [connection strings](#)). App settings and connection strings are stored encrypted in Azure, and they're decrypted only before being injected into your app's process memory when the app starts. The encryption keys are rotated regularly.

Alternatively, you can integrate your App Service app with [Azure Key Vault](#) for advanced secrets management. By [accessing the Key Vault with a managed identity](#), your App Service app can securely access the secrets you need.

Network isolation

Except for the **Isolated** pricing tier, all tiers run your apps on the shared network infrastructure in App Service. For example, the public IP addresses and front-end load balancers are shared with other tenants. The **Isolated** tier gives you complete network isolation by running your apps inside a dedicated [App Service environment](#). An App Service environment runs in your own instance of [Azure Virtual Network](#). It lets you:

- Serve your apps through a dedicated public endpoint, with dedicated front ends.
- Serve internal application using an internal load balancer (ILB), which allows access only from inside your Azure Virtual Network. The ILB has an IP address from your private subnet, which provides total isolation of your apps from the internet.
- [Use an ILB behind a web application firewall \(WAF\)](#). The WAF offers enterprise-level protection to your public-facing applications, such as DDoS protection, URI filtering, and SQL injection prevention.

For more information, see [Introduction to Azure App Service Environments](#).

App Service networking features

2/26/2020 • 15 minutes to read • [Edit Online](#)

Applications in the Azure App Service can be deployed in multiple ways. By default, App Service hosted apps are directly internet accessible and can only reach internet hosted endpoints. Many customer applications however need to control the inbound and outbound network traffic. There are several features available in the App Service to satisfy those needs. The challenge is knowing what feature should be used to solve a given problem. This document is intended to help customers determine what feature should be used based on some example use cases.

There are two primary deployment types for the Azure App Service. There is the multi-tenant public service, which hosts App Service plans in the Free, Shared, Basic, Standard, Premium, and Premiumv2 pricing SKUs. Then there is the single tenant App Service Environment(ASE), which hosts Isolated SKU App Service plans directly in your Azure Virtual Network (VNet). The features you use will vary on if you are in the multi-tenant service or in an ASE.

Multi-tenant App Service networking features

The Azure App Service is a distributed system. The roles that handle incoming HTTP/HTTPS requests are called front-ends. The roles that host the customer workload are called workers. All of the roles in an App Service deployment exist in a multi-tenant network. Because there are many different customers in the same App Service scale unit, you cannot connect the App Service network directly to your network. Instead of connecting the networks, we need features to handle the different aspects of application communication. The features that handle requests TO your app can't be used to solve problems when making calls FROM your app. Likewise, the features that solve problems for calls FROM your app can't be used to solve problems TO your app.

INBOUND FEATURES	OUTBOUND FEATURES
App assigned address	Hybrid Connections
Access Restrictions	Gateway required VNet Integration
Service Endpoints	VNet Integration

Unless otherwise stated, all of the features can be used together. You can mix the features to solve your various problems.

Use case and features

For any given use case, there can be a few ways to solve the problem. The right feature to use is sometimes due to reasons beyond just the use case itself. The following inbound use cases suggest how to use App Service networking features to solve problems around controlling traffic going to your app.

INBOUND USE CASES	FEATURE
Support IP-based SSL needs for your app	app assigned address
Not shared, dedicated inbound address for your app	app assigned address
Restrict access to your app from a set of well-defined addresses	Access Restrictions

INBOUND USE CASES	FEATURE
Expose my app on private IPs in my VNet	ILB ASE Application Gateway with service endpoints
Restrict access to my app from resources in a VNet	Service Endpoints ILB ASE
Expose my app on a private IP in my VNet	ILB ASE private IP for inbound on an Application Gateway with service endpoints
Protect my app with a WAF	Application Gateway + ILB ASE Application Gateway with service endpoints Azure Front Door with Access Restrictions
Load balance traffic to my apps in different regions	Azure Front Door with Access Restrictions
Load balance traffic in the same region	Application Gateway with service endpoints

The following outbound use cases suggest how to use App Service networking features to solve outbound access needs for your app.

OUTBOUND USE CASES	FEATURE
Access resources in an Azure Virtual Network in the same region	VNet Integration ASE
Access resources in an Azure Virtual Network in a different region	Gateway required VNet Integration ASE and VNet peering
Access resources secured with service endpoints	VNet Integration ASE
Access resources in a private network not connected to Azure	Hybrid Connections
Access resources across ExpressRoute circuits	VNet Integration ASE
Secure outbound traffic from your web app	VNet Integration and Network Security Groups ASE
Route outbound traffic from your web app	VNet Integration and Route Tables ASE

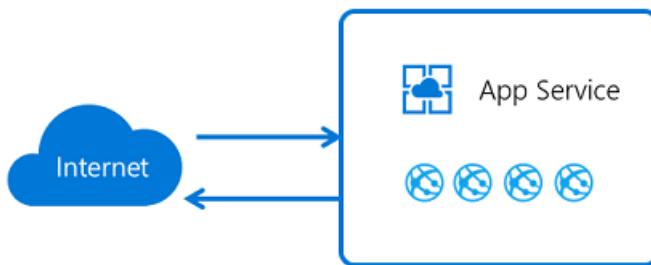
Default networking behavior

The Azure App Service scale units support many customers in each deployment. The Free and Shared SKU plans host customer workloads on multi-tenant workers. The Basic, and above plans host customer workloads that are dedicated to only one App Service plan (ASP). If you had a Standard App Service plan, then all of the apps in that plan will run on the same worker. If you scale out the worker, then all of the apps in that ASP will be replicated on a new worker for each instance in your ASP. The workers that are used for Premiumv2 are different from the workers used for the other plans. Each App Service deployment has one IP address that is used for all of the inbound traffic to the apps in that App Service deployment. There are however anywhere from 4 to 11 addresses used for making outbound calls. These addresses are shared by all of the apps in that App Service deployment. The outbound addresses are different based on the different worker types. That means that the addresses used by the

Free, Shared, Basic, Standard and Premium ASPs are different than the addresses used for outbound calls from the Premiumv2 ASPs. If you look in the properties for your app, you can see the inbound and outbound addresses that are used by your app. If you need to lock down a dependency with an IP ACL, use the possibleOutboundAddresses.

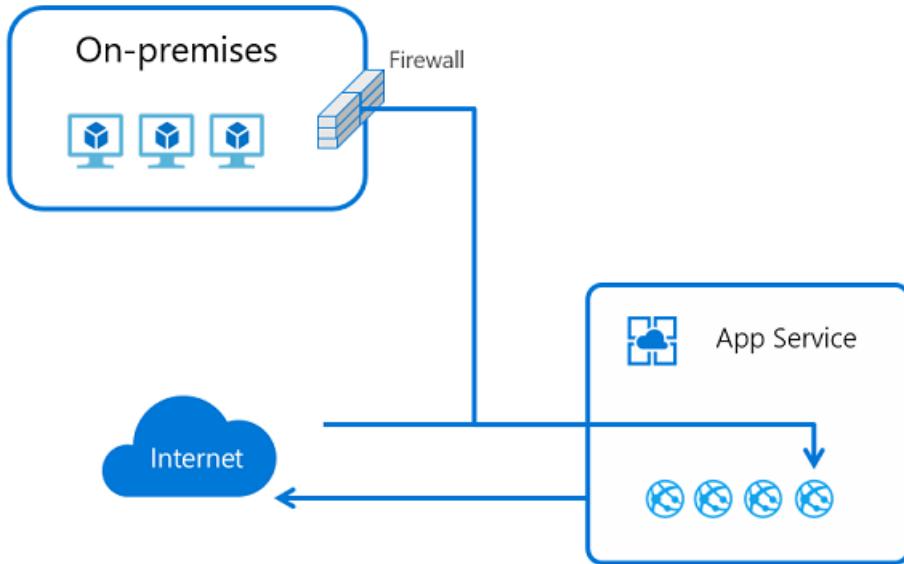
The screenshot shows the 'Properties' section of the Azure App Service configuration. Under 'Outbound IP addresses', the value is listed as '13.90.143.69,13.90.251.106,13.90.192.153,13.90.199.220,40.71.205.9'.

App Service has a number of endpoints that are used to manage the service. Those addresses are published in a separate document and are also in the AppServiceManagement IP service tag. The AppServiceManagement tag is only used with an App Service Environment (ASE) where you need to allow such traffic. The App Service inbound addresses are tracked in the AppService IP service tag. There is no IP service tag that contains the outbound addresses used by App Service.



App assigned address

The app assigned address feature is an offshoot of the IP-based SSL capability and is accessed by setting up SSL with your app. This feature can be used for IP-based SSL calls but it can also be used to give your app an address that only it has.



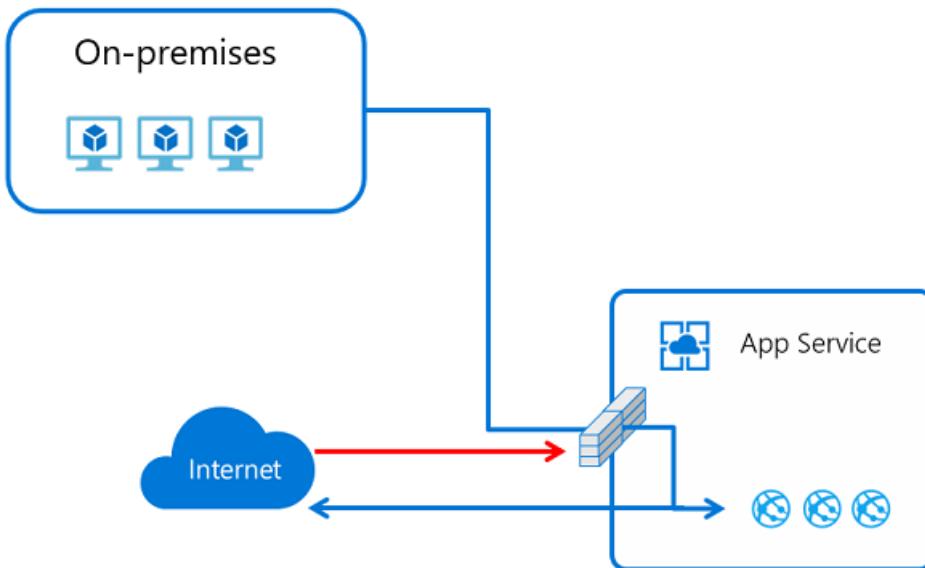
When you use an app assigned address, your traffic still goes through the same front-end roles that handle all of the incoming traffic into the App Service scale unit. The address that is assigned to your app however, is only used by your app. The use cases for this feature are to:

- Support IP-based SSL needs for your app
- Set a dedicated address for your app that is not shared with anything else

You can learn how to set an address on your app with the tutorial on [Configuring IP based SSL](#).

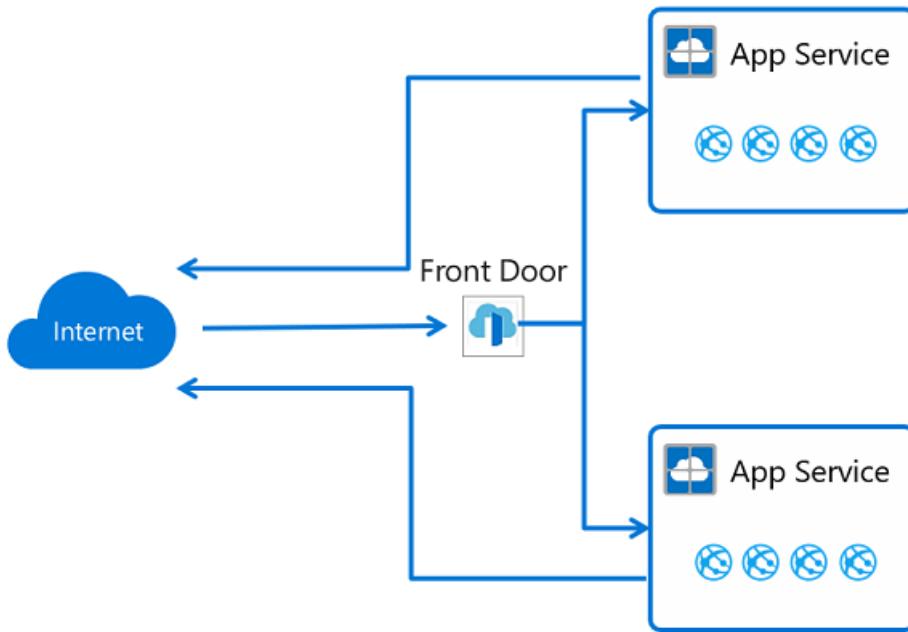
Access Restrictions

The Access Restrictions capability lets you filter **inbound** requests based on the origination IP address. The filtering action takes place on the front-end roles that are upstream from the worker roles where your apps are running. Since the front-end roles are upstream from the workers, the Access Restrictions capability can be regarded as network level protection for your apps. The feature allows you to build a list of allow and deny address blocks that are evaluated in priority order. It is similar to the Network Security Group (NSG) feature that exists in Azure Networking. You can use this feature in an ASE or in the multi-tenant service. When used with an ILB ASE, you can restrict access from private address blocks.



The Access Restrictions feature helps in scenarios where you want to restrict the IP addresses that can be used to reach your app. Among the use cases for this feature are:

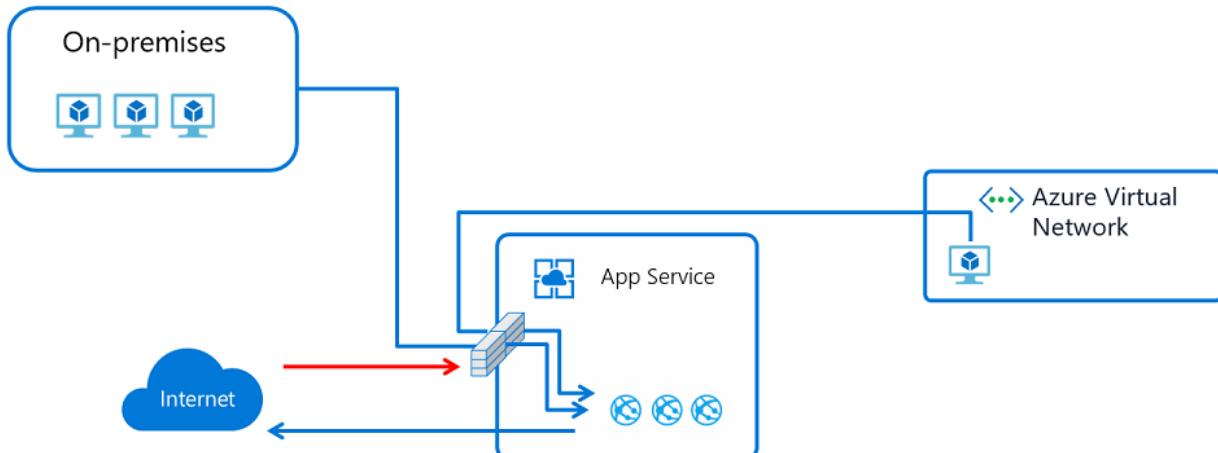
- Restrict access to your app from a set of well-defined addresses
- Restrict access to coming through a load-balancing service, such as Azure Front Door. If you wanted to lock down your inbound traffic to Azure Front Door, create rules to allow traffic from 147.243.0.0/16 and 2a01:111:2050::/44.



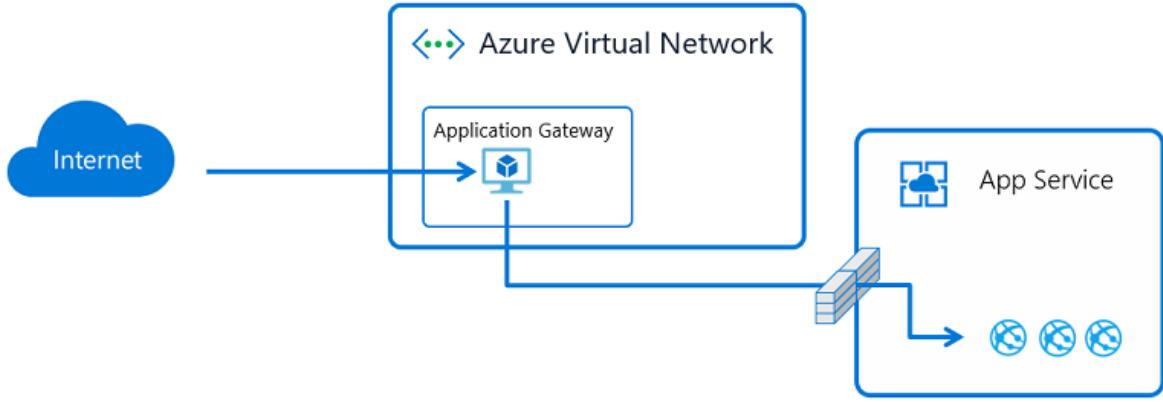
If you wish to lock down access to your app so that it can only be reached from resources in your Azure Virtual Network (VNet), you need a static public address on whatever your source is in your VNet. If the resources do not have a public address, you should use the Service Endpoints feature instead. Learn how to enable this feature with the tutorial on [Configuring Access Restrictions](#).

Service endpoints

Service endpoints allows you to lock down **inbound** access to your app such that the source address must come from a set of subnets that you select. This feature works in conjunction with the IP Access Restrictions. Service endpoints are set in the same user experience as the IP Access Restrictions. You can build an allow/deny list of access rules that includes public addresses as well as subnets in your VNets. This feature supports scenarios such as:



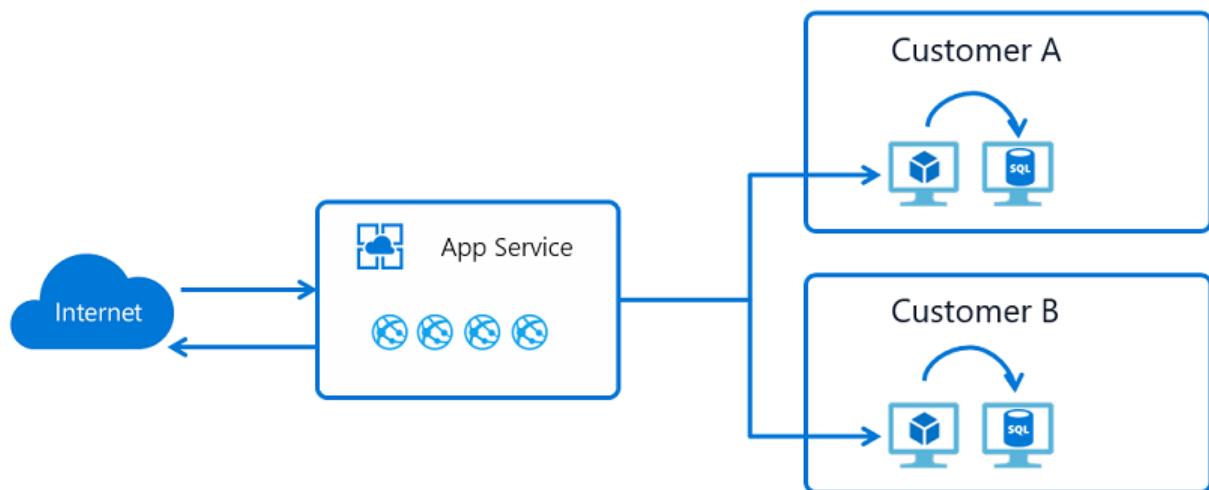
- Setting up an Application Gateway with your app to lock down inbound traffic to your app
- Restricting access to your app to resources in your VNet. This can include VMs, ASEs, or even other apps that use VNet Integration



You can learn more about configuring service endpoints with your app in the tutorial on [Configuring Service Endpoint Access Restrictions](#)

Hybrid Connections

App Service Hybrid Connections enables your apps to make **outbound** calls to specified TCP endpoints. The endpoint can be on-premises, in a VNet or anywhere that allows outbound traffic to Azure on port 443. The feature requires the installation of a relay agent called the Hybrid Connection Manager (HCM) on a Windows Server 2012 or newer host. The HCM needs to be able to reach Azure Relay at port 443. The HCM can be downloaded from the App Service Hybrid Connections UI in the portal.



The App Service Hybrid Connections feature is built on the Azure Relay Hybrid Connections capability. App Service uses a specialized form of the feature that only supports making outbound calls from your app to a TCP host and port. This host and port only need to resolve on the host where the HCM is installed. When the app, in App Service, does a DNS lookup on the host and port defined in your Hybrid Connection, the traffic is automatically redirected to go through the Hybrid Connection and out the Hybrid Connection Manager. To learn more about Hybrid Connections, read the documentation on [App Service Hybrid Connections](#)

This feature is commonly used to:

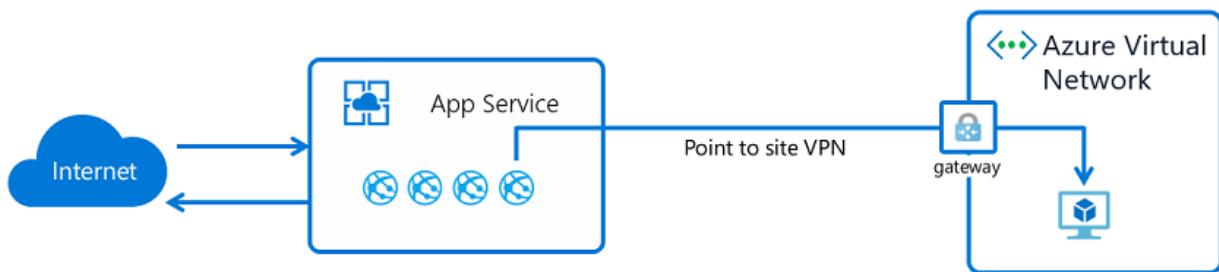
- Access resources in private networks that are not connected to Azure with a VPN or ExpressRoute
- Support lift and shift of on-premises apps to App Service without needing to also move supporting databases
- Securely provide access to a single host and port per Hybrid Connection. Most networking features open access to a network and with Hybrid Connections you only have the single host and port you can reach.
- Cover scenarios not covered by other outbound connectivity methods
- Perform development in App Service where the apps can easily leverage on-premises resources

Because the feature enables access to on-premises resources without an inbound firewall hole, it is popular with developers. The other outbound App Service networking features are very Azure Virtual Networking related. Hybrid Connections does not have a dependency on going through a VNet and can be used for a wider variety of networking needs. It is important to note that the App Service Hybrid Connections feature does not care or know what you are doing on top of it. That is to say that you can use it to access a database, a web service or an arbitrary TCP socket on a mainframe. The feature essentially tunnels TCP packets.

While Hybrid Connections is popular for development, it is also used in numerous production applications as well. It is great for accessing a web service or database, but is not appropriate for situations involving creating many connections.

Gateway required VNet Integration

The gateway required App Service VNet Integration feature enables your app to make **outbound** requests into an Azure Virtual Network. The feature works by connecting the host your app is running on to a Virtual Network gateway on your VNet with a point-to-site VPN. When you configure the feature, your app gets one of the point-to-site addresses assigned to each instance. This feature enables you to access resources in either Classic or Resource Manager VNets in any region.



This feature solves the problem of accessing resources in other VNets and can even be used to connect through a VNet to either other VNets or even on-premises. It does not work with ExpressRoute connected VNets but does with Site-to-site VPN connected networks. It is normally inappropriate to use this feature from an app in an App Service Environment (ASE), because the ASE is already in your VNet. The use cases that this feature solves are:

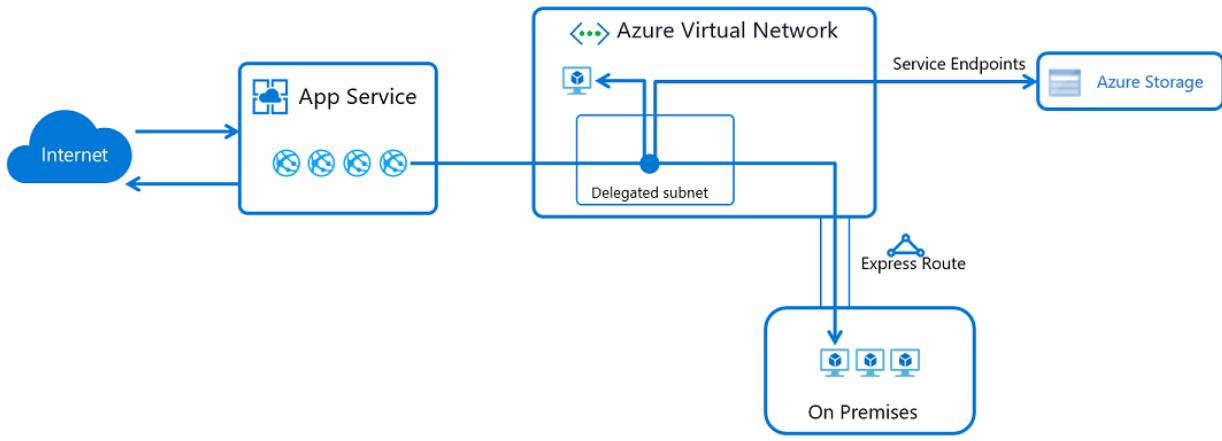
- Accessing resources on private IPs in your Azure virtual networks
- Accessing resources on-premises if there is a site-to-site VPN
- Accessing resources in peered VNets

When this feature is enabled, your app will use the DNS server that the destination VNet is configured with. You can read more on this feature in the documentation on [App Service VNet Integration](#).

VNet Integration

The gateway required VNet Integration feature is very useful but still does not solve accessing resources across ExpressRoute. On top of needing to reach across ExpressRoute connections, there is a need for apps to be able to make calls to service endpoint secured services. To solve both of those additional needs, another VNet Integration capability was added. The new VNet Integration feature enables you to place the backend of your app in a subnet in a Resource Manager VNet in the same region. This feature is not available from an App Service Environment, which is already in a VNet. This feature enables:

- Accessing resources in Resource Manager VNets in the same region
- Accessing resources that are secured with service endpoints
- Accessing resources that are accessible across ExpressRoute or VPN connections
- Securing all outbound traffic
- Force tunneling all outbound traffic.



To learn more about this feature, read the docs on [App Service VNet Integration](#).

App Service Environment

An App Service Environment (ASE) is a single tenant deployment of the Azure App Service that runs in your VNet. The ASE enables use cases such as:

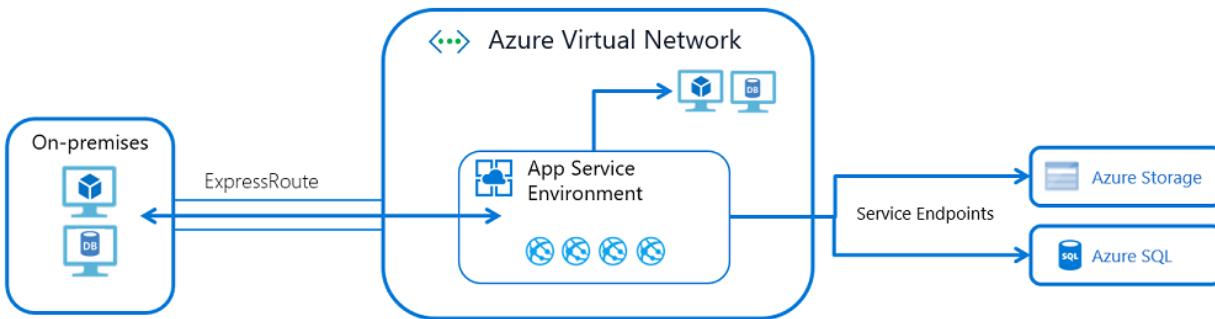
- Access resources in your VNet
- Access resources across ExpressRoute
- Expose your apps with a private address in your VNet
- Access resources across service endpoints

With an ASE, you do not need to use features like VNet Integration or service endpoints because the ASE is already in your VNet. If you want to access resources like SQL or Storage over service endpoints, enable service endpoints on the ASE subnet. If you want to access resources in the VNet, there is no additional configuration required. If you want to access resources across ExpressRoute, you are already in the VNet and do not need to configure anything on the ASE or the apps inside it.

Because the apps in an ILB ASE can be exposed on a private IP address, you can easily add WAF devices to expose just the apps that you want to the internet and keep the rest secure. It lends itself to easy development of multi-tier applications.

There are some things that are not yet possible from the multi-tenant service that are from an ASE. Those include things like:

- Expose your apps on a private IP address
- Secure all outbound traffic with network controls that are not a part of your app
- Host your apps in a single tenant service
- Scale up to many more instances than are possible in the multi-tenant service
- Load private CA client certificates for use by your apps with private CA secured endpoints
- Force TLS 1.1 across all of the apps hosted in the system without any ability to disable at the app level
- Provide a dedicated outbound address for all of the apps in your ASE that is not shared with any customers



The ASE provides the best story around isolated and dedicated app hosting but does come with some management challenges. Some things to consider before using an operational ASE are:

- An ASE runs inside your VNet but does have dependencies outside of the VNet. Those dependencies must be allowed. Read more in [Networking considerations for an App Service Environment](#)
- An ASE does not scale immediately like the multi-tenant service. You need to anticipate scaling needs rather than reactively scaling.
- An ASE does have a higher up front cost associated with it. In order to get the most out of your ASE, you should plan on putting many workloads into one ASE rather than have it used for small efforts
- The apps in an ASE cannot restrict access to some apps in an ASE and not others.
- The ASE is in a subnet and any networking rules apply to all the traffic to and from that ASE. If you want to assign inbound traffic rules for just one app, use Access Restrictions.

Combining features

The features noted for the multi-tenant service can be used together to solve more elaborate use cases. Two of the more common use cases are described here but they are just examples. By understanding what the various features do, you can solve nearly all of your system architecture needs.

Inject app into a VNet

A common request is on how to put your app in a VNet. Putting your app into a VNet means that the inbound and outbound endpoints for an app are within a VNet. The ASE provides the best solution to solve this problem but, you can get most of what is needed with in the multi-tenant service by combining features. For example, you can host intranet only applications with private inbound and outbound addresses by:

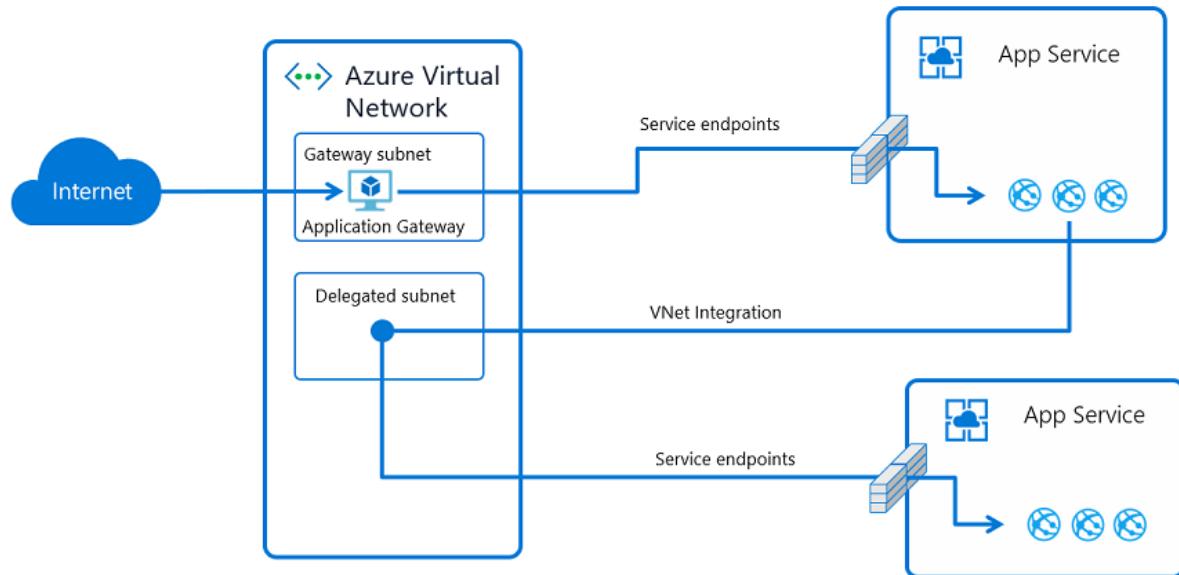
- Creating an Application Gateway with private inbound and outbound address
- Securing inbound traffic to your app with service endpoints
- Use the new VNet Integration so the backend of your app is in your VNet

This deployment style would not give you a dedicated address for outbound traffic to the internet or give you the ability to lock down all outbound traffic from your app. This deployment style would give you a much of what you would only otherwise get with an ASE.

Create multi-tier applications

A multi-tier application is an application where the API backend apps can only be accessed from the front-end tier. To create a multi-tier application, you can:

- Use VNet Integration to connect the backend of your front-end web app with a subnet in a VNet
- Use service endpoints to secure inbound traffic to your API app to only coming from the subnet used by your front-end web app



You can have multiple front-end apps use the same API app by using VNet Integration from the other front-end apps and service endpoints from the API app with their subnets.

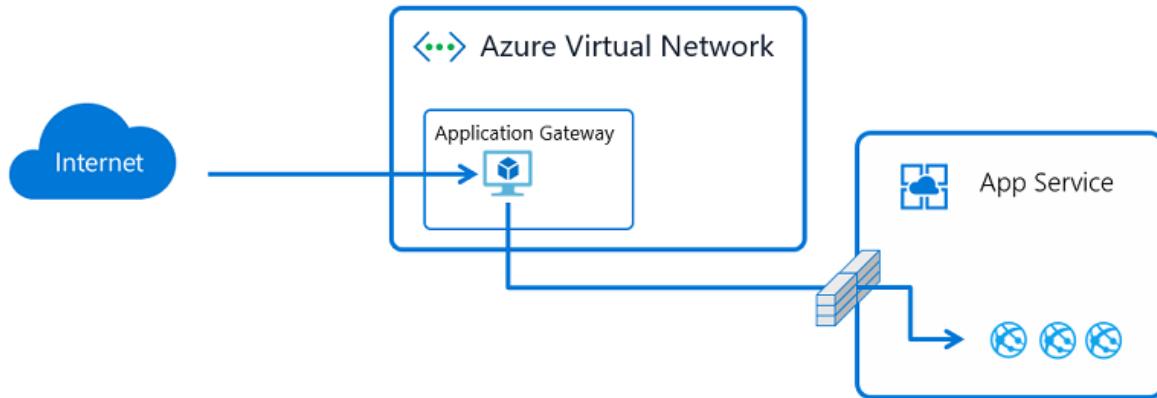
Application Gateway integration with service endpoints

12/10/2019 • 4 minutes to read • [Edit Online](#)

There are three variations of App Service that require slightly different configuration of the integration with Azure Application Gateway. The variations include regular App Service - also known as multi-tenant, Internal Load Balancer (ILB) App Service Environment (ASE) and External ASE. This article will walk through how to configure it with App Service (multi-tenant) and discuss considerations about ILB, and External ASE.

Integration with App Service (multi-tenant)

App Service (multi-tenant) has a public internet facing endpoint. Using [service endpoints](#) you can allow traffic only from a specific subnet within an Azure Virtual Network and block everything else. In the following scenario, we'll use this functionality to ensure that an App Service instance can only receive traffic from a specific Application Gateway instance.



There are two parts to this configuration besides creating the App Service and the Application Gateway. The first part is enabling service endpoints in the subnet of the Virtual Network where the Application Gateway is deployed. Service endpoints will ensure all network traffic leaving the subnet towards the App Service will be tagged with the specific subnet ID. The second part is to set an access restriction of the specific web app to ensure that only traffic tagged with this specific subnet ID is allowed. You can configure it using different tools depending on preference.

Using Azure portal

With Azure portal, you follow four steps to provision and configure the setup. If you have existing resources, you can skip the first steps.

1. Create an App Service using one of the Quickstarts in the App Service documentation, for example [.Net Core Quickstart](#)
2. Create an Application Gateway using the [portal Quickstart](#), but skip the Add backend targets section.
3. Configure [App Service as a backend in Application Gateway](#), but skip the Restrict access section.
4. Finally create the [access restriction using service endpoints](#).

You can now access the App Service through Application Gateway, but if you try to access the App Service directly, you should receive a 403 HTTP error indicating that the web site is stopped.

Error 403 - 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](#).

Using Azure Resource Manager template

The [Resource Manager deployment template](#) will provision a complete scenario. The scenario consists of an App Service instance locked down with service endpoints and access restriction to only receive traffic from Application Gateway. The template includes many Smart Defaults and unique postfixes added to the resource names for it to be simple. To override them, you'll have to clone the repo or download the template and edit it.

To apply the template you can use the Deploy to Azure button found in the description of the template, or you can use appropriate PowerShell/CLI.

Using Azure Command Line Interface

The [Azure CLI sample](#) will provision an App Service locked down with service endpoints and access restriction to only receive traffic from Application Gateway. If you only need to isolate traffic to an existing App Service from an existing Application Gateway, the following command is sufficient.

```
az webapp config access-restriction add --resource-group myRG --name myWebApp --rule-name AppGwSubnet --priority 200 --subnet mySubNetName --vnet-name myVnetName
```

In the default configuration, the command will ensure both setup of the service endpoint configuration in the subnet and the access restriction in the App Service.

Considerations for ILB ASE

ILB ASE isn't exposed to the internet and traffic between the instance and an Application Gateway is therefore already isolated to the Virtual Network. The following [how-to guide](#) configures an ILB ASE and integrates it with an Application Gateway using Azure portal.

If you want to ensure that only traffic from the Application Gateway subnet is reaching the ASE, you can configure a Network security group (NSG) which affect all web apps in the ASE. For the NSG, you are able to specify the subnet IP range and optionally the ports (80/443). Make sure you don't override the [required NSG rules](#) for ASE to function correctly.

To isolate traffic to an individual web app you'll need to use ip-based access restrictions as service endpoints will not work for ASE. The IP address should be the private IP of the Application Gateway instance.

Considerations for External ASE

External ASE has a public facing load balancer like multi-tenant App Service. Service endpoints don't work for ASE, and that's why you'll have to use ip-based access restrictions using the public IP of the Application Gateway instance. To create an External ASE using the Azure portal, you can follow this [Quickstart](#)

Considerations for kudu/scm site

The scm site, also known as kudu, is an admin site, which exists for every web app. It isn't possible to reverse proxy the scm site and you most likely also want to lock it down to individual IP addresses or a specific subnet.

If you want to use the same access restrictions as the main site, you can inherit the settings using the following command.

```
az webapp config access-restriction set --resource-group myRG --name myWebApp --use-same-restrictions-for-scm-site
```

If you want to set individual access restrictions for the scm site, you can add access restrictions using the --scm-site flag like shown below.

```
az webapp config access-restriction add --resource-group myRG --name myWebApp --scm-site --rule-name KudoAccess --priority 200 --ip-address 208.130.0.0/16
```

Next steps

For more information on the App Service Environment, see [App Service Environment documentation](#).

To further secure your web app, information about Web Application Firewall on Application Gateway can be found in the [Azure Web Application Firewall documentation](#).

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](#)

Integrate your app with an Azure Virtual Network

2/26/2020 • 23 minutes to read • [Edit Online](#)

This document describes the Azure App Service virtual network integration feature and how to set it up with apps in the [Azure App Service](#). [Azure Virtual Networks](#) (VNets) allow you to place many of your Azure resources in a non-internet routable network.

The Azure App Service has two variations.

1. The multi-tenant systems that support the full range of pricing plans except Isolated
2. The App Service Environment (ASE), which deploys into your VNet and supports Isolated pricing plan apps

This document goes through the VNet Integration feature, which is for use in the multi-tenant App Service. If your app is in [App Service Environment](#), then it's already in a VNet and doesn't require use of the VNet Integration feature to reach resources in the same VNet. For details on all of the App Service networking features, read [App Service networking features](#)

VNet Integration gives your web app access to resources in your virtual network but doesn't grant inbound private access to your web app from the VNet. Private site access refers to making your app only accessible from a private network such as from within an Azure virtual network. VNet Integration is only for making outbound calls from your app into your VNet. The VNet Integration feature behaves differently when used with VNets in the same region and with VNets in other regions. The VNet Integration feature has two variations.

1. Regional VNet Integration - When connecting to Resource Manager VNets in the same region, you must have a dedicated subnet in the VNet you are integrating with.
2. Gateway required VNet Integration - When connecting to VNets in other regions or to a Classic VNet in the same region you need a Virtual Network gateway provisioned in the target VNet.

The VNet Integration features:

- require a Standard, Premium, PremiumV2, or Elastic Premium pricing plan
- support TCP and UDP
- work with App Service apps, and Function apps

There are some things that VNet Integration doesn't support including:

- mounting a drive
- AD integration
- NetBios

Gateway required VNet Integration only provides access to resources in the target VNet or in networks connected to the target VNet with peering or VPNs. Gateway required VNet Integration doesn't enable access to resources available across ExpressRoute connections or works with service endpoints.

Regardless of the version used, VNet Integration gives your web app access to resources in your virtual network but doesn't grant inbound private access to your web app from the virtual network. Private site access refers to making your app only accessible from a private network such as from within an Azure virtual network. VNet Integration is only for making outbound calls from your app into your VNet.

Enable VNet Integration

1. Go to the Networking UI in the App Service portal. Under VNet Integration, select "[Click here to configure](#)".

2. Select **Add VNet**.

The screenshot shows the 'VNet Configuration' section of the 'VNet Integration' page. At the top, there are 'Disconnect' and 'Refresh' buttons. Below them is a 'VNet Configuration' header with a 'Securely access resources available in or through your Azure VNet. [Learn more](#)' link. A large 'Add VNet' button is prominently displayed. The 'VNet Details' section shows 'VNet NAME' and 'LOCATION' both set to 'Not Configured'. The 'VNet Address Space' section includes 'Start Address' and 'End Address' fields, both currently set to 'Not Configured'.

3. The drop-down list will contain all of the Resource Manager VNets in your subscription in the same region and below that is a list of all of the Resource Manager VNets in all other regions. Select the VNet you wish to integrate with.

The screenshot shows the 'VNet Configuration' section with a dropdown menu open over the 'Virtual Network' field. The menu has two sections: 'Same region (East US)' containing 'networking-demos-vnet (eastus)' and 'networking-peer-vnet (eastus)', and 'Other regions (requires a Virtual Network Gateway configured with Point to Site VPN)' containing a long list of VNets from various regions: 'upgrade-test-vnet (westus)', 'v2demonetwerk (westus)', 'networking-eu-vnet (northeurope)', 'crosspond-vnet (westeurope)', 'v2pshell (northcentralus)', 'VNetA (northcentralus)', 'VNetB (northcentralus)', 'VNetC (northcentralus)', 'v2pshell2 (northcentralus)', 'templatetest-vnet (southcentralus)', 'jboss-rg-vnet (eastus2)', 'ignite-vnet (westus2)', 'new-app-create-vnet (westus2)', 'testasev1-vnet (westus2)', and 'eufwtest-vnet (francecentral)'. An 'OK' button is at the bottom of the dropdown.

- If the VNet is in the same region, then either create a new subnet or pick an empty pre-existing subnet.
- To select a VNet in another region, you must have a Virtual Network gateway provisioned with point to site enabled.
- To integrate with a Classic VNet, instead of clicking the VNet drop down, select **Click here to**

connect to a Classic VNet. Select the desired Classic VNet. The target VNet must already have a Virtual Network gateway provisioned with point to site enabled.

The screenshot shows the Azure portal interface for configuring VNet integration. On the left, there's a navigation bar with 'Dashboard > vnet-integration-app - Networking > VNet Integration'. Below it, a 'VNet Integration' section has a 'vnet-integration-app' entry with 'Disconnect' and 'Refresh' buttons. A 'VNet Configuration' section below it says 'Securely access resources available in or through your Azure VNet. Learn more' and includes a '+ Add VNet' button. To the right, a 'Virtual Network' pane lists three VNets: 'ilbase-vnet (Classic) Central US', 'v2vnetintdemoapp-Network (Classic) East Asia', and 'networking-classic-demo (Classic) East US'. The 'ilbase-vnet' entry is highlighted with a red border.

During the integration, your app is restarted. When integration is completed, you will see details on the VNet you are integrated with.

Once your app is integrated with your VNet, it will use the same DNS server that your VNet is configured with, unless it is Azure DNS Private Zones. You currently cannot use VNet Integration with Azure DNS Private Zones.

Regional VNet Integration

Using regional VNet Integration enables your app to access:

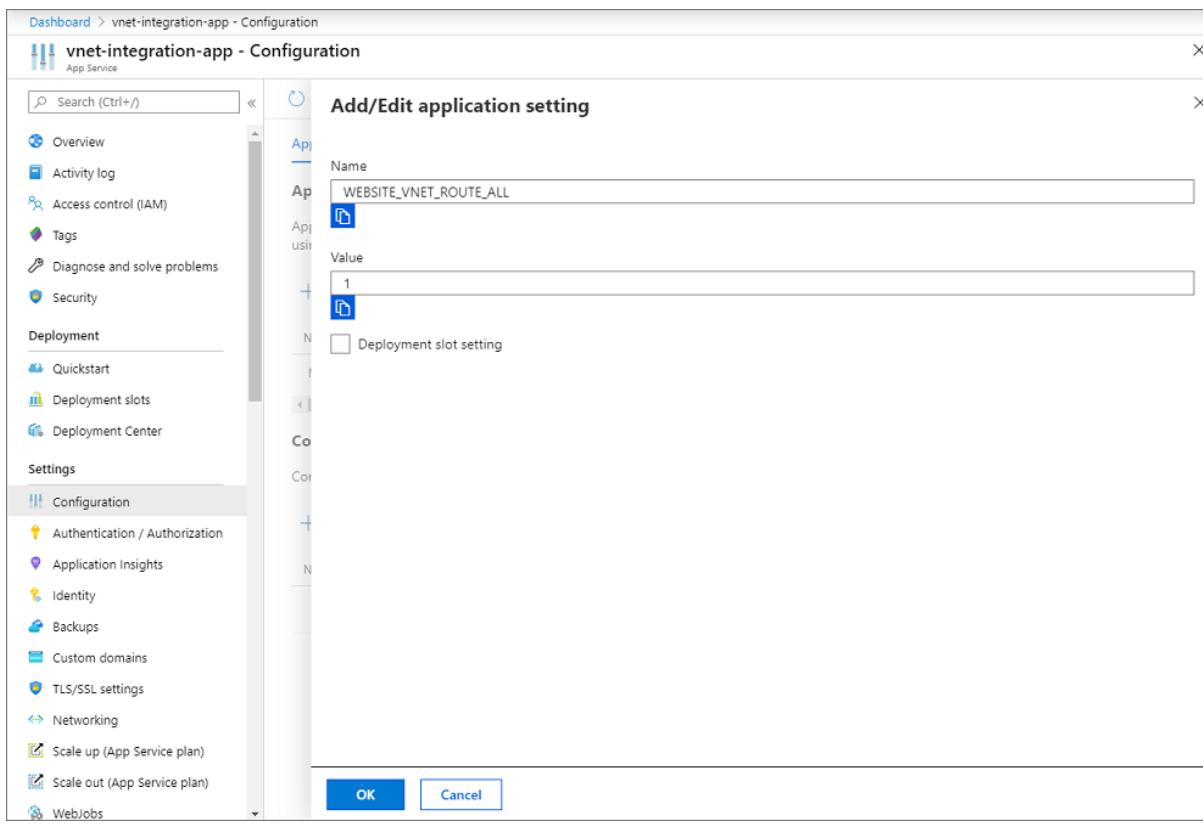
- resources in the VNet in the same region that you integrate with
- resources in VNets peered to your VNet that are in the same region
- service endpoint secured services
- resources across ExpressRoute connections
- resources in the VNet you are connected to
- resources across peered connections including ExpressRoute connections
- private endpoints

When using VNet Integration with VNets in the same region, you can use the following Azure Networking features:

- Network Security Groups(NSGs) - You can block outbound traffic with a Network Security Group that is placed on your integration subnet. The inbound rules do not apply as you cannot use VNet Integration to provide inbound access to your web app.
- Route Tables (UDRs) - You can place a route table on the integration subnet to send outbound traffic where you want.

By default, your app will only route RFC1918 traffic into your VNet. If you want to route all of your outbound traffic into your VNet, apply the app setting WEBSITE_VNET_ROUTE_ALL to your app. To configure the app setting:

1. Go to the Configuration UI in your app portal. Select **New application setting**
2. Type **WEBSITE_VNET_ROUTE_ALL** in the Name field and **1** in the Value field



3. Select **OK**

4. Select **Save**

If you route all of your outbound traffic into your VNet, then it will be subject to the NSGs and UDRs that are applied to your integration subnet. When you route all of your outbound traffic into your VNet, your outbound addresses will still be the outbound addresses that are listed in your app properties unless you provide routes to send the traffic elsewhere.

There are some limitations with using VNet Integration with VNets in the same region:

- You cannot reach resources across global peering connections
- The feature is only available from newer App Service scale units that support PremiumV2 App Service plans.
- The integration subnet can only be used by only one App Service plan
- The feature cannot be used by Isolated plan apps that are in an App Service Environment
- The feature requires an unused subnet that is a /27 with 32 addresses or larger in a Resource Manager VNet
- The app and the VNet must be in the same region
- You cannot delete a VNet with an integrated app. Remove the integration before deleting the VNet.
- You can only integrate with VNets in the same subscription as the web app
- You can have only one regional VNet Integration per App Service plan. Multiple apps in the same App Service plan can use the same VNet.
- You cannot change the subscription of an app or an App Service plan while there is an app that is using Regional VNet Integration

One address is used for each App Service plan instance. If you scale your app to five instances, then five addresses are used. Since subnet size cannot be changed after assignment, you must use a subnet that is large enough to accommodate whatever scale your app may reach. A /26 with 64 addresses is the recommended size. A /26 with 64 addresses will accommodate a Premium App Service plan with 30 instances. When you scale an App Service plan up or down, you need twice as many addresses for a short period of time.

If you want your apps in another App Service plan to reach a VNet that is connected to already by apps in another App Service plan, you need to select a different subnet than the one being used by the pre-existing VNet

Integration.

The feature is in preview for Linux. The Linux form of the feature only supports making calls to RFC 1918 addresses (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16).

Web App for Containers

If you use App Service on Linux with the built-in images, regional VNet Integration works without additional changes. If you use Web App for Containers, you need to modify your docker image in order to use VNet Integration. In your docker image, use the PORT environment variable as the main web server's listening port, instead of using a hardcoded port number. The PORT environment variable is automatically set by App Service platform at the container startup time. If you are using SSH, then the SSH daemon must be configured to listen on the port number specified by the SSH_PORT environment variable when using regional VNet integration. There is no support for gateway required VNet Integration on Linux.

Service Endpoints

Regional VNet Integration enables you to use service endpoints. To use service endpoints with your app, use regional VNet Integration to connect to a selected VNet and then configure service endpoints on the subnet you used for the integration.

Network Security Groups

Network Security Groups enable you to block inbound and outbound traffic to resources in a VNet. A web app using regional VNet Integration can use [Network Security Group](#) to block outbound traffic to resources in your VNet or the internet. To block traffic to public addresses, you must have the application setting WEBSITE_VNET_ROUTE_ALL set to 1. The inbound rules in an NSG do not apply to your app as VNet Integration only affects outbound traffic from your app. To control inbound traffic to your web app, use the Access Restrictions feature. An NSG that is applied to your integration subnet will be in effect regardless of any routes applied to your integration subnet. If WEBSITE_VNET_ROUTE_ALL was set to 1 and you did not have any routes affecting public address traffic on your integration subnet, all of your outbound traffic would still be subject to NSGs assigned to your integration subnet. If WEBSITE_VNET_ROUTE_ALL was not set, NSGs would only be applied to RFC1918 traffic.

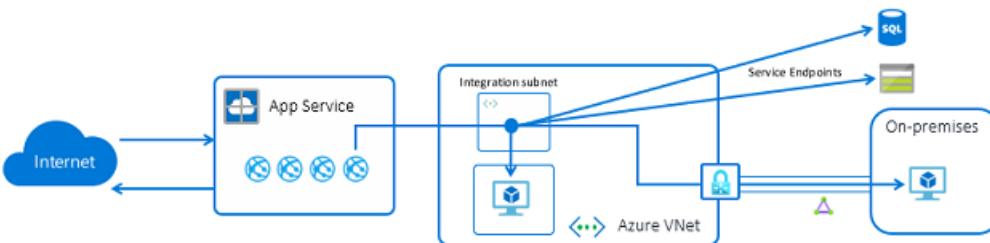
Routes

Route Tables enable you to route outbound traffic from your app to wherever you want. By default, route tables will only affect your RFC1918 destination traffic. If you set WEBSITE_VNET_ROUTE_ALL to 1, then all of your outbound calls will be affected. Routes that are set on your integration subnet will not affect replies to inbound app requests. Common destinations can include firewall devices or gateways. If you want to route all outbound traffic on-premises, you can use a route table to send all outbound traffic to your ExpressRoute gateway. If you do route traffic to a gateway, be sure to set routes in the external network to send any replies back.

Border Gateway Protocol (BGP) routes will also affect your app traffic. If you have BGP routes from something like an ExpressRoute gateway, your app outbound traffic will be affected. By default, BGP routes will only affect your RFC1918 destination traffic. If WEBSITE_VNET_ROUTE_ALL is set to 1, then all outbound traffic can be affected by your BGP routes.

How Regional VNet Integration works

Apps in the App Service are hosted on worker roles. The Basic and higher pricing plans are dedicated hosting plans where there are no other customers workloads running on the same workers. Regional VNet Integration works by mounting virtual interfaces with addresses in the delegated subnet. Because the from address is in your VNet, it can access to most things in or through your VNet just like a VM in your VNet would. The networking implementation is different than running a VM in your VNet and that is why some networking features are not yet available while using this feature.



When regional VNet Integration is enabled, your app will still make outbound calls to the internet through the same channels as normal. The outbound addresses that are listed in the app properties portal are still the addresses used by your app. What changes for your app are, calls to service endpoint secured services or RFC 1918 addresses goes into your VNet. If WEBSITE_VNET_ROUTE_ALL is set to 1, then all outbound traffic can be sent into your VNet.

The feature only supports one virtual interface per worker. One virtual interface per worker means one regional VNet Integration per App Service plan. All of the apps in the same App Service plan can use the same VNet Integration but if you need an app to connect to an additional VNet, you will need to create another App Service plan. The virtual interface used is not a resource that customers have direct access to.

Due to the nature of how this technology operates, the traffic that is used with VNet Integration does not show up in Network Watcher or NSG flow logs.

Gateway required VNet Integration

Gateway required VNet Integration supports connecting to a VNet in another region, or to a Classic VNet.

Gateway required VNet Integration:

- Enables an app to connect to only 1 VNet at a time
- Enables up to five VNets to be integrated within an App Service Plan
- Allows the same VNet to be used by multiple apps in an App Service Plan without impacting the total number that can be used by an App Service plan. If you have six apps using the same VNet in the same App Service plan, that counts as 1 VNet being used.
- Supports a 99.9% SLA due to the SLA on the gateway
- Enables your apps to use the DNS that the VNet is configured with
- Requires a Virtual Network route-based gateway configured with SSTP point-to-site VPN before it can be connected to app.

You can't use gateway required VNet Integration:

- With Linux apps
- With a VNet connected with ExpressRoute
- To access Service Endpoints secured resources
- With a coexistence gateway that supports both ExpressRoute and point to site/site to site VPNs

Set up a gateway in your VNet

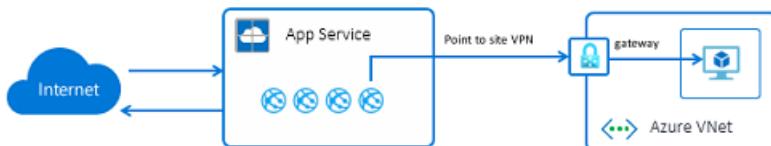
To create a gateway:

1. [Create a gateway subnet](#) in your VNet.
2. [Create the VPN gateway](#). Select a route-based VPN type.
3. [Set the point to site addresses](#). If the gateway isn't in the basic SKU, then IKEV2 must be disabled in the point to site configuration and SSTP must be selected. The point to site address space must be in the RFC 1918 address blocks, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16

If you are just creating the gateway for use with App Service VNet Integration, then you do not need to upload a certificate. Creating the gateway can take 30 minutes. You will not be able to integrate your app with your VNet until the gateway is provisioned.

How gateway required VNet Integration works

Gateway required VNet Integration built on top of point to site VPN technology. Point to site VPNs limits network access to just the virtual machine hosting the app. Apps are restricted to only send traffic out to the internet, through Hybrid Connections or through VNet Integration. When your app is configured with the portal to use gateway required VNet Integration, a complex negotiation is managed on your behalf to create and assign certificates on the gateway and the application side. The end result is that the workers used to host your apps are able to directly connect to the virtual network gateway in the selected VNet.



Accessing on-premises resources

Apps can access on-premises resources by integrating with VNets that have site-to-site connections. If you are using the gateway required VNet Integration, you need to update your on-premises VPN gateway routes with your point-to-site address blocks. When the site-to-site VPN is first set up, the scripts used to configure it should set up routes properly. If you add the point-to-site addresses after you create your site-to-site VPN, you need to update the routes manually. Details on how to do that vary per gateway and are not described here. You cannot have BGP configured with a site-to-site VPN connection.

There is no additional configuration required for the regional VNet Integration feature to reach through your VNet, and to on-premises. You simply need to connect your VNet to on-premises using ExpressRoute or a site-to-site VPN.

NOTE

The gateway required VNet Integration feature doesn't integrate an app with a VNet that has an ExpressRoute Gateway. Even if the ExpressRoute Gateway is configured in [coexistence mode](#) the VNet Integration doesn't work. If you need to access resources through an ExpressRoute connection, then you can use the regional VNet Integration feature or an [App Service Environment](#), which runs in your VNet.

Peering

If you are using peering with the regional VNet Integration, you do not need to do any additional configuration.

If you are using the gateway required VNet Integration with peering, you need to configure a few additional items. To configure peering to work with your app:

1. Add a peering connection on the VNet your app connects to. When adding the peering connection, enable **Allow virtual network access** and check **Allow forwarded traffic** and **Allow gateway transit**.
2. Add a peering connection on the VNet that is being peered to the VNet you are connected to. When adding the peering connection on the destination VNet, enable **Allow virtual network access** and check **Allow forwarded traffic** and **Allow remote gateways**.
3. Go to the App Service plan > Networking > VNet Integration UI in the portal. Select the VNet your app connects to. Under the routing section, add the address range of the VNet that is peered with the VNet your app is connected to.

Managing VNet Integration

Connecting and disconnecting with a VNet is at an app level. Operations that can affect the VNet Integration across multiple apps are at the App Service plan level. From the app > Networking > VNet Integration portal, you can get details on your VNet. You can see similar information at the ASP level in the ASP > Networking > VNet Integration portal.

The only operation you can take in the app view of your VNet Integration is to disconnect your app from the VNet it is currently connected to. To disconnect your app from a VNet, select **Disconnect**. Your app will be restarted when you disconnect from a VNet. Disconnecting doesn't change your VNet. The subnet or gateway is not removed. If you then want to delete your VNet, you need to first disconnect your app from the VNet and delete the resources in it such as gateways.

The ASP VNet Integration UI will show you all of the VNet integrations used by the apps in your ASP. To see details on each VNet, click on the VNet you are interested in. There are two actions you can perform here for gateway required VNet Integration.

- **Sync network.** The sync network operation is only for the gateway-dependent VNet Integration feature. Performing a sync network operation ensures that your certificates and network information are in sync. If you add or change the DNS of your VNet, you need to perform a **Sync network** operation. This operation will restart any apps using this VNet.
- **Add routes** Adding routes will drive outbound traffic into your VNet.

Gateway required VNet Integration Routing The routes that are defined in your VNet are used to direct traffic into your VNet from your app. If you need to send additional outbound traffic into the VNet, then you can add those address blocks here. This capability only works with gateway required VNet Integration. Route tables do not affect your app traffic when using gateway required VNet Integration the way that they do with regional VNet Integration.

Gateway required VNet Integration Certificates When the gateway required VNet Integration enabled, there is a required exchange of certificates to ensure the security of the connection. Along with the certificates are the DNS configuration, routes, and other similar things that describe the network. If certificates or network information is changed, you need to click "Sync Network". When you click "Sync Network", you cause a brief outage in connectivity between your app and your VNet. While your app isn't restarted, the loss of connectivity could cause your site to not function properly.

Pricing details

The regional VNet Integration feature has no additional charge for use beyond the ASP pricing tier charges.

There are three related charges to the use of the gateway required VNet Integration feature:

- ASP pricing tier charges - Your apps need to be in a Standard, Premium, or PremiumV2 App Service Plan. You can see more details on those costs here: [App Service Pricing](#).
- Data transfer costs - There is a charge for data egress, even if the VNet is in the same data center. Those charges are described in [Data Transfer Pricing Details](#).
- VPN Gateway costs - There is a cost to the VNet gateway that is required for the point-to-site VPN. The details are on the [VPN Gateway Pricing](#) page.

Troubleshooting

While the feature is easy to set up, that doesn't mean that your experience will be problem free. Should you encounter problems accessing your desired endpoint there are some utilities you can use to test connectivity from the app console. There are two consoles that you can use. One is the Kudu console and the other is the console in the Azure portal. To reach the Kudu console from your app, go to Tools -> Kudu. You can also reach the Kudu console at [sitename].scm.azurewebsites.net. Once the website loads, go to the Debug console tab. To get to the Azure portal hosted console then from your app go to Tools -> Console.

Tools

The tools **ping**, **nslookup**, and **tracert** won't work through the console due to security constraints. To fill the void, two separate tools added. In order to test DNS functionality, we added a tool named nameresolver.exe. The syntax is:

```
nameresolver.exe hostname [optional: DNS Server]
```

You can use **nameresolver** to check the hostnames that your app depends on. This way you can test if you have anything mis-configured with your DNS or perhaps don't have access to your DNS server. You can see the DNS server that your app will use in the console by looking at the environmental variables WEBSITE_DNS_SERVER and WEBSITE_DNS_ALT_SERVER.

The next tool allows you to test for TCP connectivity to a host and port combination. This tool is called **tcpping** and the syntax is:

```
tcpping.exe hostname [optional: port]
```

The **tcpping** utility tells you if you can reach a specific host and port. It only can show success if: there is an application listening at the host and port combination, and there is network access from your app to the specified host and port.

Debugging access to VNet hosted resources

There are a number of things that can prevent your app from reaching a specific host and port. Most of the time it is one of three things:

- **A firewall is in the way.** If you have a firewall in the way, you will hit the TCP timeout. The TCP timeout is 21 seconds in this case. Use the **tcpping** tool to test connectivity. TCP timeouts can be due to many things beyond firewalls but start there.
- **DNS isn't accessible.** The DNS timeout is three seconds per DNS server. If you have two DNS servers, the timeout is 6 seconds. Use nameresolver to see if DNS is working. Remember you can't use nslookup as that doesn't use the DNS your VNet is configured with. If inaccessible, you could have a firewall or NSG blocking access to DNS or it could be down.

If those items don't answer your problems, look first for things like:

regional VNet Integration

- is your destination a non-RFC1918 address and you do not have WEBSITE_VNET_ROUTE_ALL set to 1
- is there an NSG blocking egress from your integration subnet
- if going across ExpressRoute or a VPN, is your on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your VNet but not on-premises, check your routes.
- do you have enough permissions to set delegation on the integration subnet? During regional VNet Integration configuration, your integration subnet will be delegated to Microsoft.Web. The VNet Integration UI will delegate the subnet to Microsoft.Web automatically. If your account does not have sufficient networking permissions to set delegation, you will need someone who can set attributes on your integration subnet to delegate the subnet. To manually delegate the integration subnet, go to the Azure Virtual Network subnet UI and set delegation for Microsoft.Web.

gateway required VNet Integration

- is the point-to-site address range in the RFC 1918 ranges (10.0.0.0-10.255.255.255 / 172.16.0.0-172.31.255.255 / 192.168.0.0-192.168.255.255)?
- Does the Gateway show as being up in the portal? If your gateway is down, then bring it back up.
- Do certificates show as being in sync or do you suspect that the network configuration was changed? If your

certificates are out of sync or you suspect that there has been a change made to your VNet configuration that wasn't synced with your ASPs, then hit "Sync Network".

- if going across a VPN, is the on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your VNet but not on-premises, check your routes.
- are you trying to use a coexistence gateway that supports both point to site and ExpressRoute? Coexistence gateways are not supported with VNet Integration

Debugging networking issues is a challenge because there you cannot see what is blocking access to a specific host:port combination. Some of the causes include:

- you have a firewall up on your host preventing access to the application port from your point to site IP range. Crossing subnets often requires Public access.
- your target host is down
- your application is down
- you had the wrong IP or hostname
- your application is listening on a different port than what you expected. You can match your process ID with the listening port by using "netstat -aon" on the endpoint host.
- your network security groups are configured in such a manner that they prevent access to your application host and port from your point to site IP range

Remember that you don't know what address your app will actually use. It could be any address in the integration subnet or point-to-site address range, so you need to allow access from the entire address range.

Additional debug steps include:

- connect to a VM in your VNet and attempt to reach your resource host:port from there. To test for TCP access, use the PowerShell command **test-netconnection**. The syntax is:

```
test-netconnection hostname [optional: -Port]
```

- bring up an application on a VM and test access to that host and port from the console from your app using **tcpping**

On-premises resources

If your app cannot reach a resource on-premises, then check if you can reach the resource from your VNet. Use the **test-netconnection** PowerShell command to check for TCP access. If your VM can't reach your on-premises resource, your VPN or ExpressRoute connection may not be configured properly.

If your VNet hosted VM can reach your on-premises system but your app can't, then the cause is likely one of the following reasons:

- your routes are not configured with your subnet or point to site address ranges in your on-premises gateway
- your network security groups are blocking access for your Point-to-Site IP range
- your on-premises firewalls are blocking traffic from your Point-to-Site IP range
- you are trying to reach a non-RFC 1918 address using the regional VNet Integration feature

Automation

There is CLI support for regional VNet Integration. To access to the following commands, [Install Azure CLI](#).

```
az webapp vnet-integration --help

Group
  az webapp vnet-integration : Methods that list, add, and remove virtual network integrations
    from a webapp.
    This command group is in preview. It may be changed/removed in a future release.
Commands:
  add    : Add a regional virtual network integration to a webapp.
  list   : List the virtual network integrations on a webapp.
  remove : Remove a regional virtual network integration from webapp.
```

```
az appservice vnet-integration --help
```

```
Group
  az appservice vnet-integration : A method that lists the virtual network integrations used in an
    appservice plan.
    This command group is in preview. It may be changed/removed in a future release.
Commands:
  list : List the virtual network integrations used in an appservice plan.
```

For gateway required VNet Integration, you can integrate App Service with an Azure Virtual Network using PowerShell. For a ready-to-run script, see [Connect an app in Azure App Service to an Azure Virtual Network](#).

Azure App Service Hybrid Connections

12/2/2019 • 10 minutes to read • [Edit Online](#)

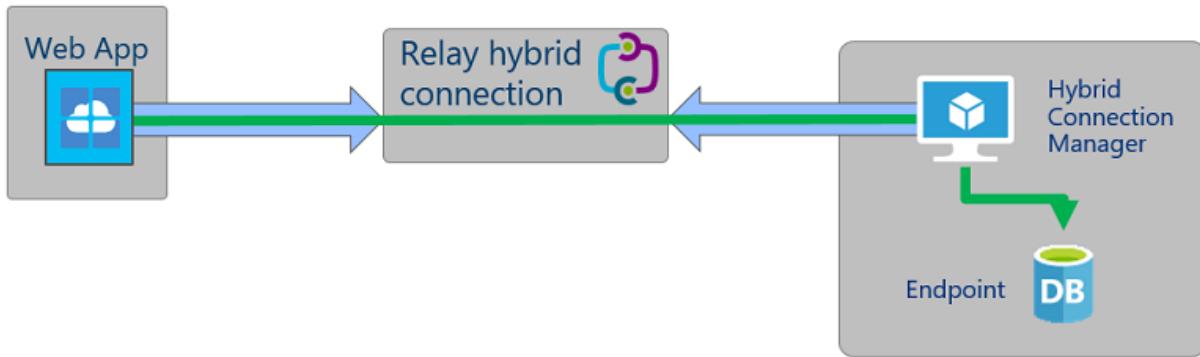
Hybrid Connections is both a service in Azure and a feature in Azure App Service. As a service, it has uses and capabilities beyond those that are used in App Service. To learn more about Hybrid Connections and their usage outside App Service, see [Azure Relay Hybrid Connections](#).

Within App Service, Hybrid Connections can be used to access application resources in other networks. It provides access from your app to an application endpoint. It does not enable an alternate capability to access your application. As used in App Service, each Hybrid Connection correlates to a single TCP host and port combination. This means that the Hybrid Connection endpoint can be on any operating system and any application, provided you are accessing a TCP listening port. The Hybrid Connections feature does not know or care what the application protocol is, or what you are accessing. It is simply providing network access.

How it works

The Hybrid Connections feature consists of two outbound calls to Azure Service Bus Relay. There is a connection from a library on the host where your app is running in App Service. There is also a connection from the Hybrid Connection Manager (HCM) to Service Bus Relay. The HCM is a relay service that you deploy within the network hosting the resource you are trying to access.

Through the two joined connections, your app has a TCP tunnel to a fixed host:port combination on the other side of the HCM. The connection uses TLS 1.2 for security and shared access signature (SAS) keys for authentication and authorization.



When your app makes a DNS request that matches a configured Hybrid Connection endpoint, the outbound TCP traffic will be redirected through the Hybrid Connection.

NOTE

This means that you should try to always use a DNS name for your Hybrid Connection. Some client software does not do a DNS lookup if the endpoint uses an IP address instead.

App Service Hybrid Connection benefits

There are a number of benefits to the Hybrid Connections capability, including:

- Apps can access on-premises systems and services securely.
- The feature does not require an internet-accessible endpoint.

- It is quick and easy to set up.
- Each Hybrid Connection matches to a single host:port combination, helpful for security.
- It normally does not require firewall holes. The connections are all outbound over standard web ports.
- Because the feature is network level, it is agnostic to the language used by your app and the technology used by the endpoint.
- It can be used to provide access in multiple networks from a single app.

Things you cannot do with Hybrid Connections

Things you cannot do with Hybrid Connections include:

- Mount a drive.
- Use UDP.
- Access TCP-based services that use dynamic ports, such as FTP Passive Mode or Extended Passive Mode.
- Support LDAP, because it can require UDP.
- Support Active Directory, because you cannot domain join an App Service worker.

Prerequisites

- Windows App service is required. It is only available in Windows.

Add and Create Hybrid Connections in your app

To create a Hybrid Connection, go to the [Azure portal](#) and select your app. Select **Networking > Configure your Hybrid Connection endpoints**. Here you can see the Hybrid Connections that are configured for your app.

The screenshot shows the 'Hybrid connections' blade in the Azure portal. At the top, it displays the app name 'hdemoplapp'. Below that is a summary section with a circular progress bar indicating connection usage: 8 active, 17 pending, and 0/25 total. There are buttons to 'Download connection manager' and 'Add hybrid connection'. A table below lists 'Classic hybrid connections' with columns for NAME, STATUS, HOST, and PORT. Both rows show 'No results'.

To add a new Hybrid Connection, select **[+] Add hybrid connection**. You'll see a list of the Hybrid Connections that you already created. To add one or more of them to your app, select the ones you want, and then select **Add selected Hybrid Connection**.

The screenshot shows two windows side-by-side. The left window is titled 'Hybrid connections' and displays a summary of hybrid connection usage. It includes a section for 'App service plan (pricing tier): hcdemoplans (Standard)', a 'Connections used' summary (8 active, 17 pending, 25 total), and a location set to 'North Central US'. Below this are sections for 'Download connection manager' and 'Add hybrid connection' (with a blue button). The right window is titled 'Add hybrid connection' and contains instructions: 'To use a hybrid connection with your app select it from the list and click on 'Add selected hybrid connection'. Click on 'Create new hybrid connection' if you need to create a new one.' It lists a single hybrid connection entry:

NAME	HOST	PORT	NAMESPACE	LOCATION
workstation-mysql	compy-wkstn	3306	workstation-mysql	North Central US

If you want to create a new Hybrid Connection, select **Create new hybrid connection**. Specify the:

- Hybrid Connection name.
- Endpoint hostname.
- Endpoint port.
- Service Bus namespace you want to use.

Create new hybrid connection

* Endpoint Name ?
Enter a name

* Endpoint Host ?
Enter a host name

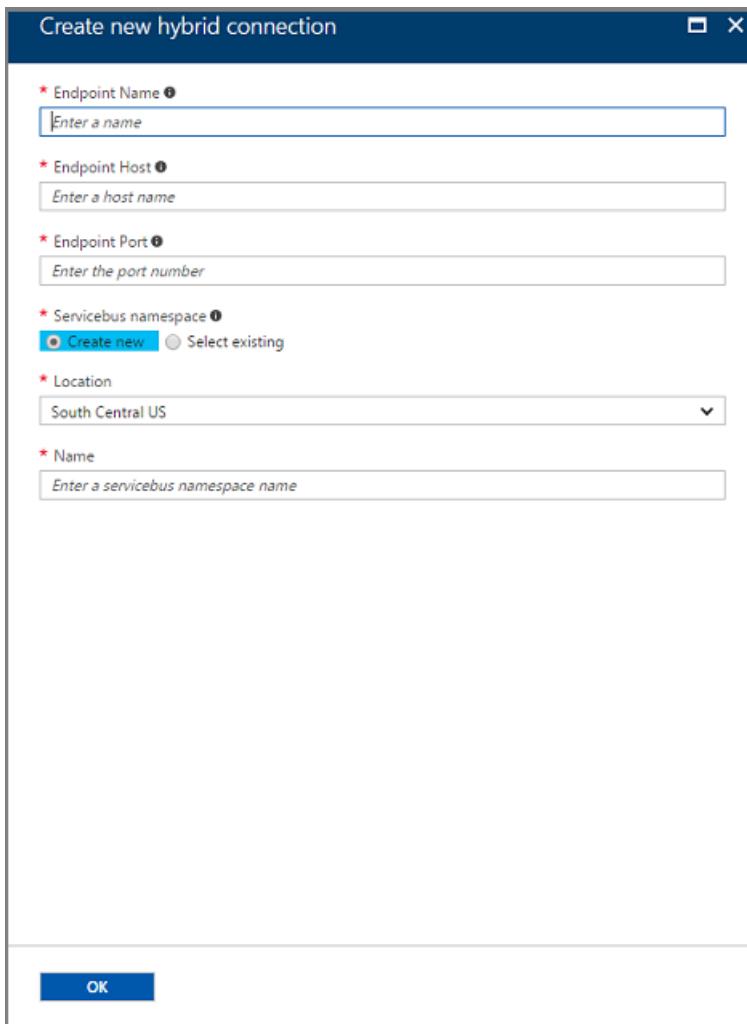
* Endpoint Port ?
Enter the port number

* Servicebus namespace ?
 Create new Select existing

* Location
South Central US

* Name
Enter a servicebus namespace name

OK



Every Hybrid Connection is tied to a Service Bus namespace, and each Service Bus namespace is in an Azure region. It's important to try to use a Service Bus namespace in the same region as your app, to avoid network induced latency.

If you want to remove your Hybrid Connection from your app, right-click it and select **Disconnect**.

When a Hybrid Connection is added to your app, you can see details on it simply by selecting it.

Hybrid connections

hcdemoapp1

Refresh

Hybrid connections

App Service integration with hybrid connections enables your app to access a single TCP endpoint per hybrid connection. Here you can manage the new and classic hybrid connections used by your app.

App service plan (pricing tier): hcdeplan (Standard)

Location: North Central US

Connections used: 8 / 25

Download connection manager

Add hybrid connection

NAME	STATUS	ENDPO...	NAMES...
workst...	Conne...	compy...	workst...

Classic hybrid connections

Add classic hybrid connection

Properties

hcdemoapp1

ENDPOINT NAME: workstation-mysql

ENDPOINT HOST: compy-wkstrn

ENDPOINT PORT: 3306

SERVICE BUS NAMESPACE: workstation-mysql

NAMESPACE LOCATION: North Central US

HYBRID CONNECTION MANAGERS: 2 connected

GATEWAY CONNECTION STRING: Endpoint=sb://purple-north-relay.servicebus.windows.net/

Create a Hybrid Connection in the Azure Relay portal

In addition to the portal experience from within your app, you can create Hybrid Connections from within the Azure Relay portal. For a Hybrid Connection to be used by App Service, it must:

- Require client authorization.
- Have a metadata item, named endpoint, that contains a host:port combination as the value.

Hybrid Connections and App Service plans

App Service Hybrid Connections are only available in Basic, Standard, Premium, and Isolated pricing SKUs. There are limits tied to the pricing plan.

PRICING PLAN	NUMBER OF HYBRID CONNECTIONS USABLE IN THE PLAN
Basic	5
Standard	25
Premium	200
Isolated	200

The App Service plan UI shows you how many Hybrid Connections are being used and by what apps.

The screenshot shows two adjacent windows from the Azure portal. The left window, titled 'Hybrid connections' (hcdemoplan), lists a single entry: 'workstation-mysql' with an endpoint of '3306 : compy-wkstn'. The right window, titled 'Properties' (hcdemoplan), displays detailed information for this connection. Key details include:

- Endpoint Name: workstation-mysql
- Endpoint Host: compy-wkstn
- Endpoint Port: 3306
- Service Bus Namespace: workstation-mysql (highlighted with a blue border)
- Namespace Location: North Central US
- Connection Manager Connections: 2
- Number of Connected Sites: 1
- Gateway Connection String: Endpoint=sb://purple-north-relay.servicebus.windows.net/ (with a copy icon)

Select the Hybrid Connection to see details. You can see all the information that you saw at the app view. You can also see how many other apps in the same plan are using that Hybrid Connection.

There is a limit on the number of Hybrid Connection endpoints that can be used in an App Service plan. Each Hybrid Connection used, however, can be used across any number of apps in that plan. For example, a single Hybrid Connection that is used in five separate apps in an App Service plan counts as one Hybrid Connection.

Pricing

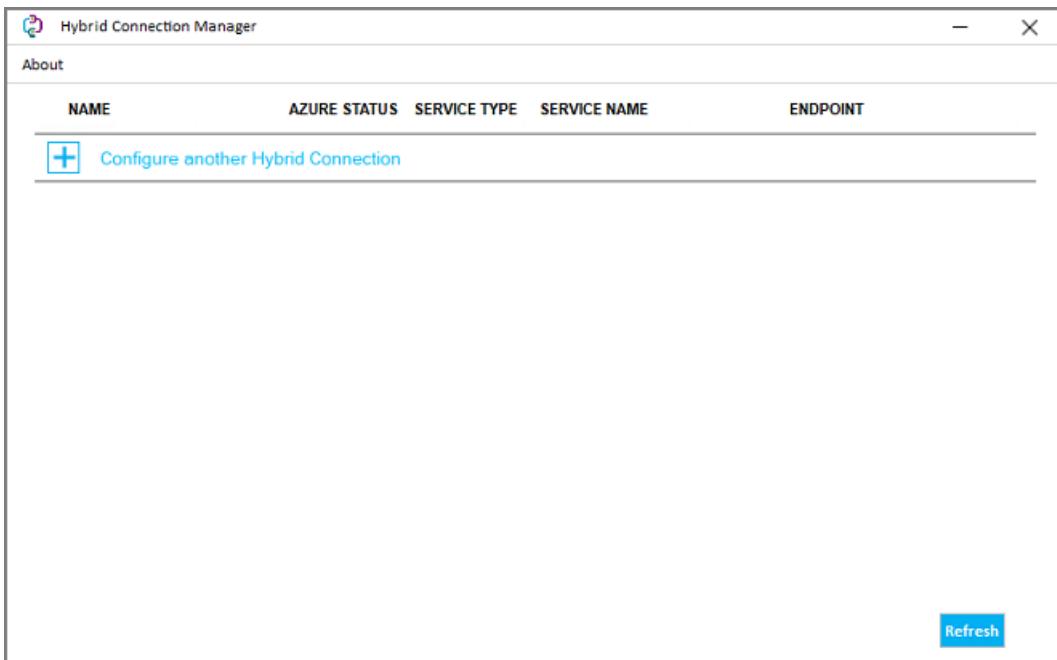
In addition to there being an App Service plan SKU requirement, there is an additional cost to using Hybrid Connections. There is a charge for each listener used by a Hybrid Connection. The listener is the Hybrid Connection Manager. If you had five Hybrid Connections supported by two Hybrid Connection Managers, that would be 10 listeners. For more information, see [Service Bus pricing](#).

Hybrid Connection Manager

The Hybrid Connections feature requires a relay agent in the network that hosts your Hybrid Connection endpoint. That relay agent is called the Hybrid Connection Manager (HCM). To download HCM, from your app in the [Azure portal](#), select **Networking > Configure your Hybrid Connection endpoints**.

This tool runs on Windows Server 2012 and later. The HCM runs as a service and connects outbound to Azure Relay on port 443.

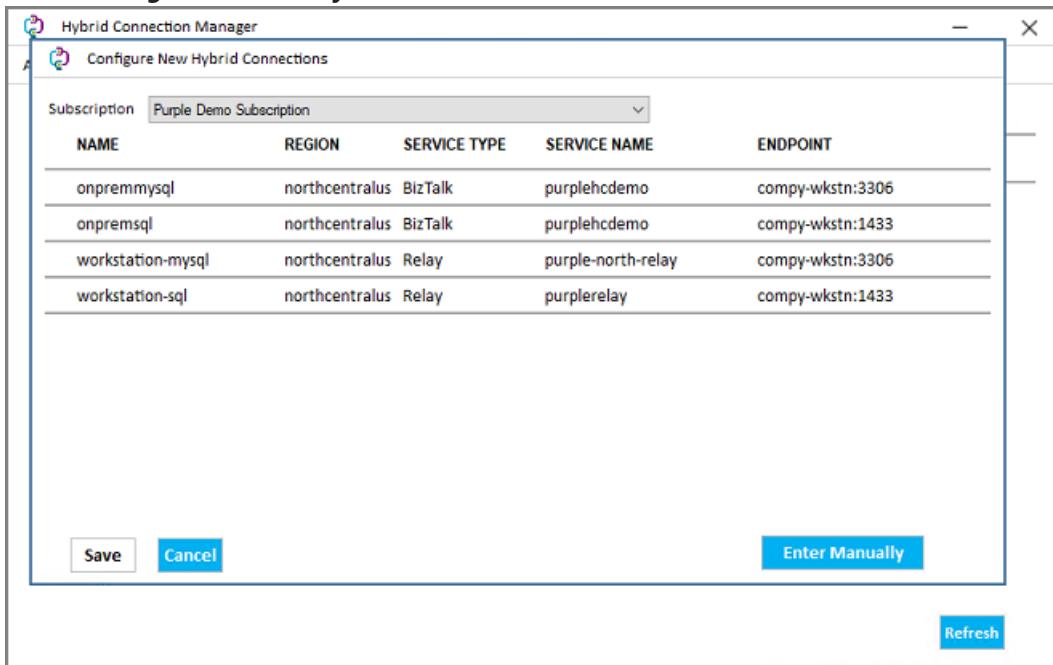
After installing HCM, you can run HybridConnectionManagerUi.exe to use the UI for the tool. This file is in the Hybrid Connection Manager installation directory. In Windows 10, you can also just search for *Hybrid Connection Manager UI* in your search box.



When you start the HCM UI, the first thing you see is a table that lists all the Hybrid Connections that are configured with this instance of the HCM. If you want to make any changes, first authenticate with Azure.

To add one or more Hybrid Connections to your HCM:

1. Start the HCM UI.
2. Select **Configure another Hybrid Connection**.



3. Sign in with your Azure account to get your Hybrid Connections available with your subscriptions. The HCM does not continue to use your Azure account beyond that.
4. Choose a subscription.
5. Select the Hybrid Connections that you want the HCM to relay.

The screenshot shows the 'Hybrid Connection Manager' window. At the top, there's a toolbar with icons for 'About', 'Configure another Hybrid Connection', and a refresh button. Below the toolbar is a table with the following columns: NAME, AZURE STATUS, SERVICE TYPE, SERVICE NAME, and ENDPOINT. A row is present with the values: 'workstation-mysql', 'Connected', 'Relay', 'purple-north-relay', and 'compy-wkstn:3306'. At the bottom right of the main area is a 'Refresh' button.

6. Select **Save**.

You can now see the Hybrid Connections you added. You can also select the configured Hybrid Connection to see details.

The screenshot shows the 'Hybrid Connection Details' dialog box. It contains a 'Remove' link and a table with the following data:

Name	workstation-mysql
Namespace	purple-north-relay
Endpoint	compy-wkstn:3306
Status	Connected
Service Bus Endpoint	purple-north-relay.servicebus.windows.net
Azure IP Address	<IP address>
Azure Ports	80, 443
Created On	4/12/2017 3:24:59 AM
Last Updated	4/12/2017 3:25:17 AM

At the bottom right of the dialog is a 'Close' button.

To support the Hybrid Connections it is configured with, HCM requires:

- TCP access to Azure over port 443.
- TCP access to the Hybrid Connection endpoint.
- The ability to do DNS look-ups on the endpoint host and the Service Bus namespace.

NOTE

Azure Relay relies on Web Sockets for connectivity. This capability is only available on Windows Server 2012 or later. Because of that, HCM is not supported on anything earlier than Windows Server 2012.

Redundancy

Each HCM can support multiple Hybrid Connections. Also, any given Hybrid Connection can be supported by

multiple HCMs. The default behavior is to route traffic across the configured HCMs for any given endpoint. If you want high availability on your Hybrid Connections from your network, run multiple HCMs on separate machines. The load distribution algorithm used by the Relay service to distribute traffic to the HCMs is random assignment.

Manually add a Hybrid Connection

To enable someone outside your subscription to host an HCM instance for a given Hybrid Connection, share the gateway connection string for the Hybrid Connection with them. You can see the gateway connection string in the Hybrid Connection properties in the [Azure portal](#). To use that string, select **Enter Manually** in the HCM, and paste in the gateway connection string.



Upgrade

There are periodic updates to the Hybrid Connection Manager to fix issues or provide improvements. When upgrades are released, a popup will show up in the HCM UI. Applying the upgrade will apply the changes and restart the HCM.

Adding a Hybrid Connection to your app programmatically

The APIs noted below can be used directly to manage the Hybrid Connections connected to your apps.

```
/subscriptions/[subscription name]/resourceGroups/[resource group name]/providers/Microsoft.Web/sites/[app name]/hybridConnectionNamespaces/[relay namespace name]/relays/[hybrid connection name]?api-version=2016-08-01
```

The JSON object associated with a Hybrid Connection looks like:

```
{
  "name": "[hybrid connection name]",
  "type": "Microsoft.Relay/Namespaces/HybridConnections",
  "location": "[location]",
  "properties": {
    "serviceBusNamespace": "[namespace name]",
    "relayName": "[hybrid connection name]",
    "relayArmUri": "/subscriptions/[subscription id]/resourceGroups/[resource group name]/providers/Microsoft.Relay/namespaces/[namespace name]/hybridconnections/[hybrid connection name]",
    "hostName": "[endpoint host name]",
    "port": [port],
    "sendKeyName": "defaultSender",
    "sendKeyValue": "[send key]"
  }
}
```

One way to use this information is with the `armclient`, which you can get from the [ARMClient](#) GitHub project. Here is an example on attaching a pre-existing Hybrid Connection to your app. Create a JSON file per the above schema like:

```
{
  "name": "relay-demo-hc",
  "type": "Microsoft.Relay/Namespaces/HybridConnections",
  "location": "North Central US",
  "properties": {
    "serviceBusNamespace": "demo-relay",
    "relayName": "relay-demo-hc",
    "relayArmUri": "/subscriptions/ebcidic-ascii-anna-nath-rak111111/resourceGroups/myrelay-
rg/providers/Microsoft.Relay/namespaces/demo-relay/hybridconnections/relay-demo-hc",
    "hostName": "my-wkstn.home",
    "port": 1433,
    "sendKeyName": "defaultSender",
    "sendKeyValue": "Th9is3is8a82lot93of3774stu887ff122235="
  }
}
```

To use this API, you need the send key and relay resource ID. If you saved your information with the filename `hctest.json`, issue this command to attach your Hybrid Connection to your app:

```
armclient login
armclient put /subscriptions/ebcidic-ascii-anna-nath-rak111111/resourceGroups/myapp-
rg/providers/Microsoft.Web/sites/myhcdemoapp/hybridConnectionNamespaces/demo-relay/relays/relay-demo-hc?api-
version=2016-08-01 @hctest.json
```

Secure your Hybrid Connections

An existing Hybrid Connection can be added to other App Service Web Apps by any user who has sufficient permissions on the underlying Azure Service Bus Relay. This means that if you must prevent others from reusing that same Hybrid Connection (for example when the target resource is a service that does not have any additional security measures in place to prevent unauthorized access), you must lock down access to the Azure Service Bus Relay.

Anyone with `Reader` access to the Relay will be able to see the Hybrid Connection when attempting to add it to their Web App in the Azure Portal, but they will not be able to *add* it as they lack the permissions to retrieve the connection string which is used to establish the relay connection. In order to successfully add the Hybrid Connection, they must have the `listKeys` permission (

`Microsoft.Relay/namespaces/hybridConnections/authorizationRules/listKeys/action`). The `Contributor` role or any other role which includes this permission on the Relay will allow users to use the Hybrid Connection and add it to their own Web Apps.

Troubleshooting

The status of "Connected" means that at least one HCM is configured with that Hybrid Connection, and is able to reach Azure. If the status for your Hybrid Connection does not say **Connected**, your Hybrid Connection is not configured on any HCM that has access to Azure.

The primary reason that clients cannot connect to their endpoint is because the endpoint was specified by using an IP address instead of a DNS name. If your app cannot reach the desired endpoint and you used an IP address, switch to using a DNS name that is valid on the host where the HCM is running. Also check that the DNS name resolves properly on the host where the HCM is running. Confirm that there is connectivity from the host where the HCM is running to the Hybrid Connection endpoint.

In App Service, the **tcpping** command line tool can be invoked from the Advanced Tools (Kudu) console. This tool can tell you if you have access to a TCP endpoint, but it does not tell you if you have access to a Hybrid Connection endpoint. When you use the tool in the console against a Hybrid Connection endpoint, you are only confirming that it uses a host:port combination.

If you have a command line client for your endpoint, you can test connectivity from the app console. For example, you can test access to web server endpoints by using curl.

BizTalk Hybrid Connections

The early form of this feature was called BizTalk Hybrid Connections. This capability went End of Life on May 31, 2018 and ceased operations. BizTalk hybrid connections have been removed from all apps and are not accessible through the portal or API. If you still have these older connections configured in the Hybrid Connection Manager, then you will see a status of Discontinued and display an End of Life statement at the bottom.

Hybrid Connection Manager

About

NAME	AZURE STATUS	SERVICE NAME	ENDPOINT
onpremsql	Discontinued*	purplehcdemo	compy-wkstn:1433
wordpress-mysql	Connected	purplerelay	compy-wkstn:3306

Add a new Hybrid Connection

***** BizTalk Hybrid Connections reached "end-of-life" on May 31, 2018 and is no longer operational.
Migrate to App Service Hybrid Connections instead as described [here](#).

Enter Manually **Refresh**

Controlling Azure App Service traffic with Azure Traffic Manager

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

NOTE

This article provides summary information for Microsoft Azure Traffic Manager as it relates to Azure App Service. More information about Azure Traffic Manager itself can be found by visiting the links at the end of this article.

Introduction

You can use Azure Traffic Manager to control how requests from web clients are distributed to apps in Azure App Service. When App Service endpoints are added to an Azure Traffic Manager profile, Azure Traffic Manager keeps track of the status of your App Service apps (running, stopped, or deleted) so that it can decide which of those endpoints should receive traffic.

Routing methods

Azure Traffic Manager uses four different routing methods. These methods are described in the following list as they pertain to Azure App Service.

- **Priority:** use a primary app for all traffic, and provide backups in case the primary or the backup apps are unavailable.
- **Weighted:** distribute traffic across a set of apps, either evenly or according to weights, which you define.
- **Performance:** when you have apps in different geographic locations, use the "closest" app in terms of the lowest network latency.
- **Geographic:** direct users to specific apps based on which geographic location their DNS query originates from.

For more information, see [Traffic Manager routing methods](#).

App Service and Traffic Manager Profiles

To configure the control of App Service app traffic, you create a profile in Azure Traffic Manager that uses one of the four load balancing methods described previously, and then add the endpoints (in this case, App Service) for which you want to control traffic to the profile. Your app status (running, stopped, or deleted) is regularly communicated to the profile so that Azure Traffic Manager can direct traffic accordingly.

When using Azure Traffic Manager with Azure, keep in mind the following points:

- For app only deployments within the same region, App Service already provides failover and round-robin functionality without regard to app mode.
- For deployments in the same region that use App Service in conjunction with another Azure cloud service, you can combine both types of endpoints to enable hybrid scenarios.
- You can only specify one App Service endpoint per region in a profile. When you select an app as an endpoint for one region, the remaining apps in that region become unavailable for selection for that profile.
- The App Service endpoints that you specify in an Azure Traffic Manager profile appears under the **Domain Names** section on the Configure page for the app in the profile, but is not configurable there.
- After you add an app to a profile, the **Site URL** on the Dashboard of the app's portal page displays the custom

domain URL of the app if you have set one up. Otherwise, it displays the Traffic Manager profile URL (for example, `contoso.trafficmanager.net`). Both the direct domain name of the app and the Traffic Manager URL are visible on the app's Configure page under the **Domain Names** section.

- Your custom domain names work as expected, but in addition to adding them to your apps, you must also configure your DNS map to point to the Traffic Manager URL. For information on how to set up a custom domain for an App Service app, see [Map an existing custom DNS name to Azure App Service](#).
- You can only add apps that are in standard or premium mode to an Azure Traffic Manager profile.

Next Steps

For a conceptual and technical overview of Azure Traffic Manager, see [Traffic Manager Overview](#).

Azure App Service Local Cache overview

1/6/2020 • 6 minutes to read • [Edit Online](#)

NOTE

Local cache is not supported in Function apps or containerized App Service apps, such as on [App Service on Linux](#).

Azure App Service content is stored on Azure Storage and is surfaced up in a durable manner as a content share. This design is intended to work with a variety of apps and has the following attributes:

- The content is shared across multiple virtual machine (VM) instances of the app.
- The content is durable and can be modified by running apps.
- Log files and diagnostic data files are available under the same shared content folder.
- Publishing new content directly updates the content folder. You can immediately view the same content through the SCM website and the running app (typically some technologies such as ASP.NET do initiate an app restart on some file changes to get the latest content).

While many apps use one or all of these features, some apps just need a high-performance, read-only content store that they can run from with high availability. These apps can benefit from a VM instance of a specific local cache.

The Azure App Service Local Cache feature provides a web role view of your content. This content is a write-but-discard cache of your storage content that is created asynchronously on-site startup. When the cache is ready, the site is switched to run against the cached content. Apps that run on Local Cache have the following benefits:

- They are immune to latencies that occur when they access content on Azure Storage.
- They are immune to the planned upgrades or unplanned downtimes and any other disruptions with Azure Storage that occur on servers that serve the content share.
- They have fewer app restarts due to storage share changes.

How the local cache changes the behavior of App Service

- *D:\home* points to the local cache, which is created on the VM instance when the app starts up. *D:\local* continues to point to the temporary VM-specific storage.
- The local cache contains a one-time copy of the */site* and */siteextensions* folders of the shared content store, at *D:\home\site* and *D:\home\siteextensions*, respectively. The files are copied to the local cache when the app starts up. The size of the two folders for each app is limited to 300 MB by default, but you can increase it up to 2 GB. If the copied files exceed the size of the local cache, App Service silently ignores local cache and read from the remote file share.
- The local cache is read-write. However, any modification is discarded when the app moves virtual machines or gets restarted. Do not use the local cache for apps that store mission-critical data in the content store.
- *D:\home\LogFiles* and *D:\home\Data* contain log files and app data. The two subfolders are stored locally on the VM instance, and are copied to the shared content store periodically. Apps can persist log files and data by writing them to these folders. However, the copy to the shared content store is best-effort, so it is possible for log files and data to be lost due to a sudden crash of a VM instance.
- **Log streaming** is affected by the best-effort copy. You could observe up to a one-minute delay in the streamed logs.
- In the shared content store, there is a change in the folder structure of the *LogFiles* and *Data* folders for apps

that use the local cache. There are now subfolders in them that follow the naming pattern of "unique identifier" + time stamp. Each of the subfolders corresponds to a VM instance where the app is running or has run.

- Other folders in *D:\home* remain in the local cache and are not copied to the shared content store.
- App deployment through any supported method publishes directly to the durable shared content store. To refresh the *D:\home\site* and *D:\home\siteextensions* folders in the local cache, the app needs to be restarted. To make the lifecycle seamless, see the information later in this article.
- The default content view of the SCM site continues to be that of the shared content store.

Enable Local Cache in App Service

You configure Local Cache by using a combination of reserved app settings. You can configure these app settings by using the following methods:

- [Azure portal](#)
- [Azure Resource Manager](#)

Configure Local Cache by using the Azure portal

You enable Local Cache on a per-web-app basis by using this app setting: `WEBSITE_LOCAL_CACHE_OPTION` = `Always`

Key	Value	Slot setting
WEBSITE_LOCAL_CACHE_OPTION	Always	<input checked="" type="checkbox"/> Slot setting
		<input type="checkbox"/> Slot setting
		<input type="checkbox"/> Slot setting

Configure Local Cache by using Azure Resource Manager

```
...
{
  "apiVersion": "2015-08-01",
  "type": "config",
  "name": "appsettings",
  "dependsOn": [
    "[resourceId('Microsoft.Web/sites/', variables('siteName'))]"
  ],
  "properties": {
    "WEBSITE_LOCAL_CACHE_OPTION": "Always",
    "WEBSITE_LOCAL_CACHE_SIZEINMB": "300"
  }
}
...
```

Change the size setting in Local Cache

By default, the local cache size is **300 MB**. This includes the /site and /siteextensions folders that are copied from the content store, as well as any locally created logs and data folders. To increase this limit, use the app setting `WEBSITE_LOCAL_CACHE_SIZEINMB`. You can increase the size up to **2 GB** (2000 MB) per app.

Best practices for using App Service Local Cache

We recommend that you use Local Cache in conjunction with the [Staging Environments](#) feature.

- Add the *sticky* app setting `WEBSITE_LOCAL_CACHE_OPTION` with the value `Always` to your **Production** slot. If you're using `WEBSITE_LOCAL_CACHE_SIZEINMB`, also add it as a sticky setting to your Production slot.
- Create a **Staging** slot and publish to your Staging slot. You typically don't set the staging slot to use Local Cache to enable a seamless build-deploy-test lifecycle for staging if you get the benefits of Local Cache for the production slot.
- Test your site against your Staging slot.
- When you are ready, issue a [swap operation](#) between your Staging and Production slots.
- Sticky settings include name and sticky to a slot. So when the Staging slot gets swapped into Production, it inherits the Local Cache app settings. The newly swapped Production slot will run against the local cache after a few minutes and will be warmed up as part of slot warmup after swap. So when the slot swap is complete, your Production slot is running against the local cache.

Frequently asked questions (FAQ)

How can I tell if Local Cache applies to my app?

If your app needs a high-performance, reliable content store, does not use the content store to write critical data at runtime, and is less than 2 GB in total size, then the answer is "yes"! To get the total size of your /site and /siteextensions folders, you can use the site extension "Azure Web Apps Disk Usage."

How can I tell if my site has switched to using Local Cache?

If you're using the Local Cache feature with Staging Environments, the swap operation does not complete until Local Cache is warmed up. To check if your site is running against Local Cache, you can check the worker process environment variable `WEBSITE_LOCALCACHE_READY`. Use the instructions on the [worker process environment variable](#) page to access the worker process environment variable on multiple instances.

I just published new changes, but my app does not seem to have them. Why?

If your app uses Local Cache, then you need to restart your site to get the latest changes. Don't want to publish changes to a production site? See the slot options in the previous best practices section.

Where are my logs?

With Local Cache, your logs and data folders do look a little different. However, the structure of your subfolders remains the same, except that the subfolders are nestled under a subfolder with the format "unique VM identifier" + time stamp.

I have Local Cache enabled, but my app still gets restarted. Why is that? I thought Local Cache helped with frequent app restarts.

Local Cache does help prevent storage-related app restarts. However, your app could still undergo restarts during planned infrastructure upgrades of the VM. The overall app restarts that you experience with Local Cache enabled should be fewer.

Does Local Cache exclude any directories from being copied to the faster local drive?

As part of the step that copies the storage content, any folder that is named repository is excluded. This helps with scenarios where your site content may contain a source control repository that may not be needed in day to day

operation of the app.

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 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 one highlighted by a red border:

- Wrong port is exposed: 5000 != 8080
- Verbose application logging is off.
- Link to Full Logs

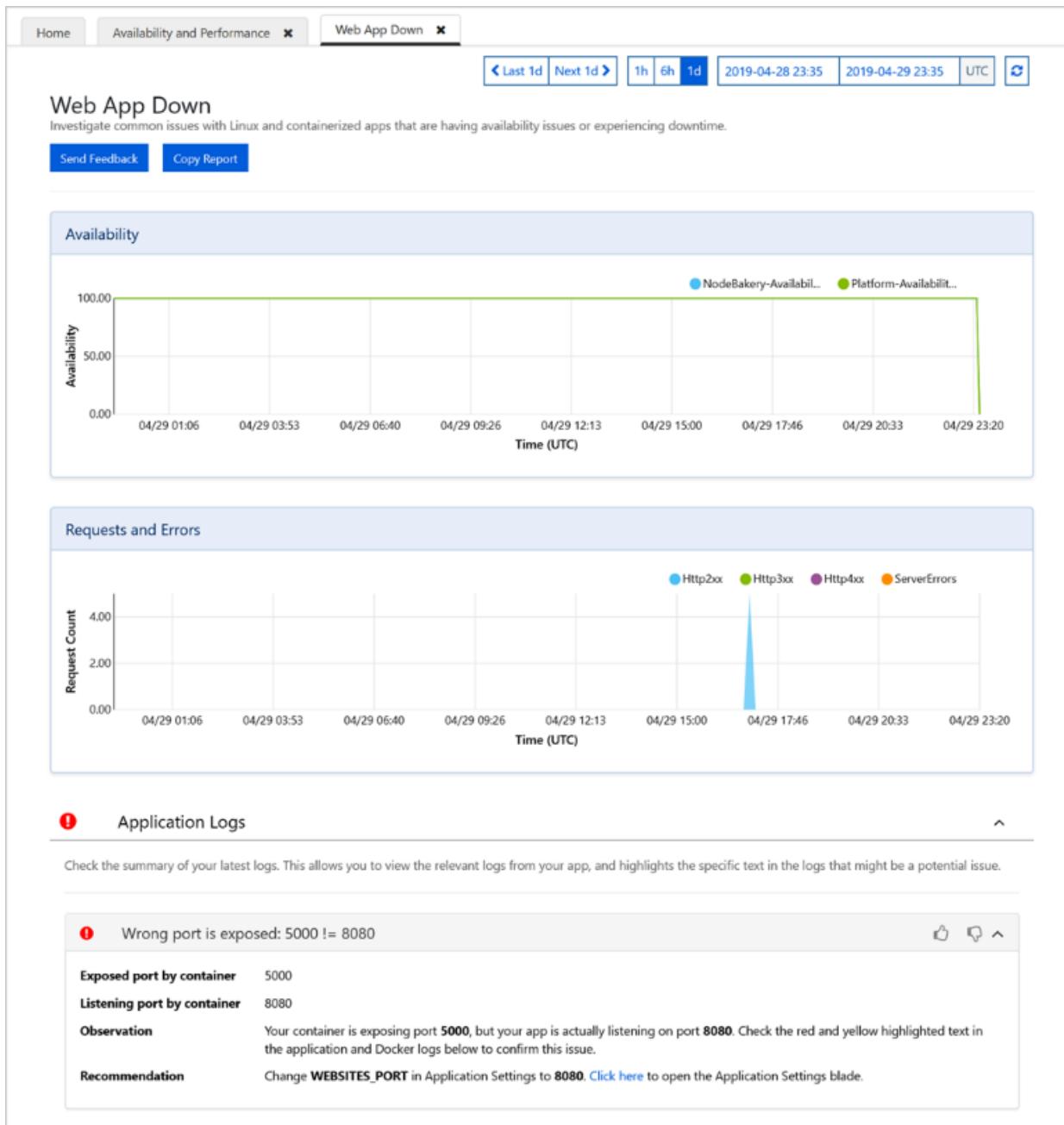
Web App Down

[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.



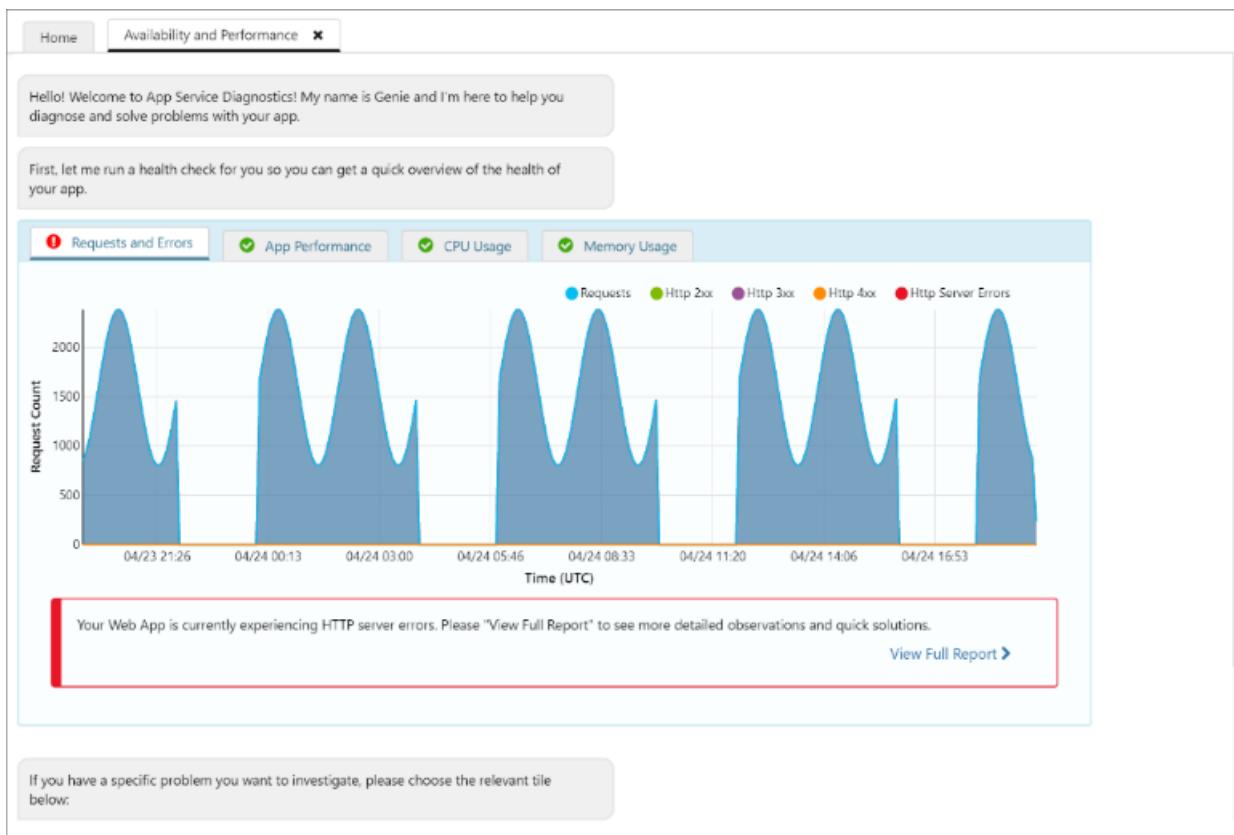
Health checkup

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
View More in App Insights		

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.

The screenshot shows the 'Troubleshooting and Next Steps' section for an 'App Service Plan'. On the left, there's a sidebar with 'Scale Out App Service Plan' and 'Mitigation' (selected), and 'Remote Profile App' with 'Investigation' (selected). The main area has a heading 'Scale out your App Service Plan' and a 'Mitigation' section. It says to increase instances to at least two. Below that is a 'Current App Service Plan' summary: S1 Standard Tier, 1 Instance Count, 2 App Count. A note says to scale out to at least two instances. At the bottom is a 'Why you should scale up' section with a bullet point about high CPU usage.

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](#).

The screenshot shows the 'Proactive CPU Monitoring' configuration page. It has a header stating '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.' Below is a section titled '- 1. Configure'.

Monitoring Enabled: A toggle switch is set to 'On'.

CPU Threshold: A slider is set to 75%.

Threshold Seconds: A slider is set to 30 sec.

Auto-healing and proactive auto-healing

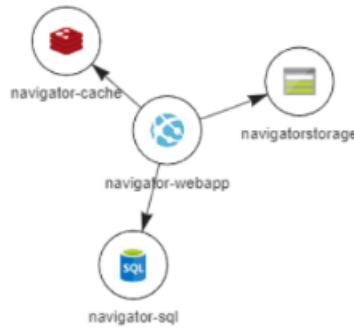
Auto-healing is a mitigation action you can take when your app is having unexpected behavior. You can set your own rules based on request count, slow request, memory limit, and HTTP status code to trigger mitigation actions. Use the tool to temporarily mitigate an unexpected behavior until you find the root cause. For more information, see [Announcing the new auto healing experience in app service diagnostics](#).

The screenshot shows the 'Configure Mitigation Rules' blade in the Azure portal. At the top, there are three tabs: 'Configure Mitigation Rules' (selected), 'ProActive Auto-Healing' (disabled), and 'View Auto-Healing History'. Below the tabs, the 'Auto-Healing Enabled' switch is set to 'On'. The main area is divided into four sections: 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), and 4. Review and Save your Settings (Current Settings: No rule configured!). At the bottom are 'Save' and 'Cancel' buttons.

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 <?xml version="1.0" encoding="utf-8"?> 2 <!-- 3 For more information on how to configure your ASP.N 4 https://go.microsoft.com/fwlink/?LinkId=301879 5 --> 6 <configuration> 7 <connectionStrings> 8 - <add name="MyDbConnection" connectionString="Server=(local);Datab 9 <add name="StorageConnection" connectionString="Data Source=(local);Datab 10 </connectionStrings> 11 <appSettings> 12 <add key="webpages:Version" value="3.0.0.0" /> 13 <add key="webpages:Enabled" value="false" /></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

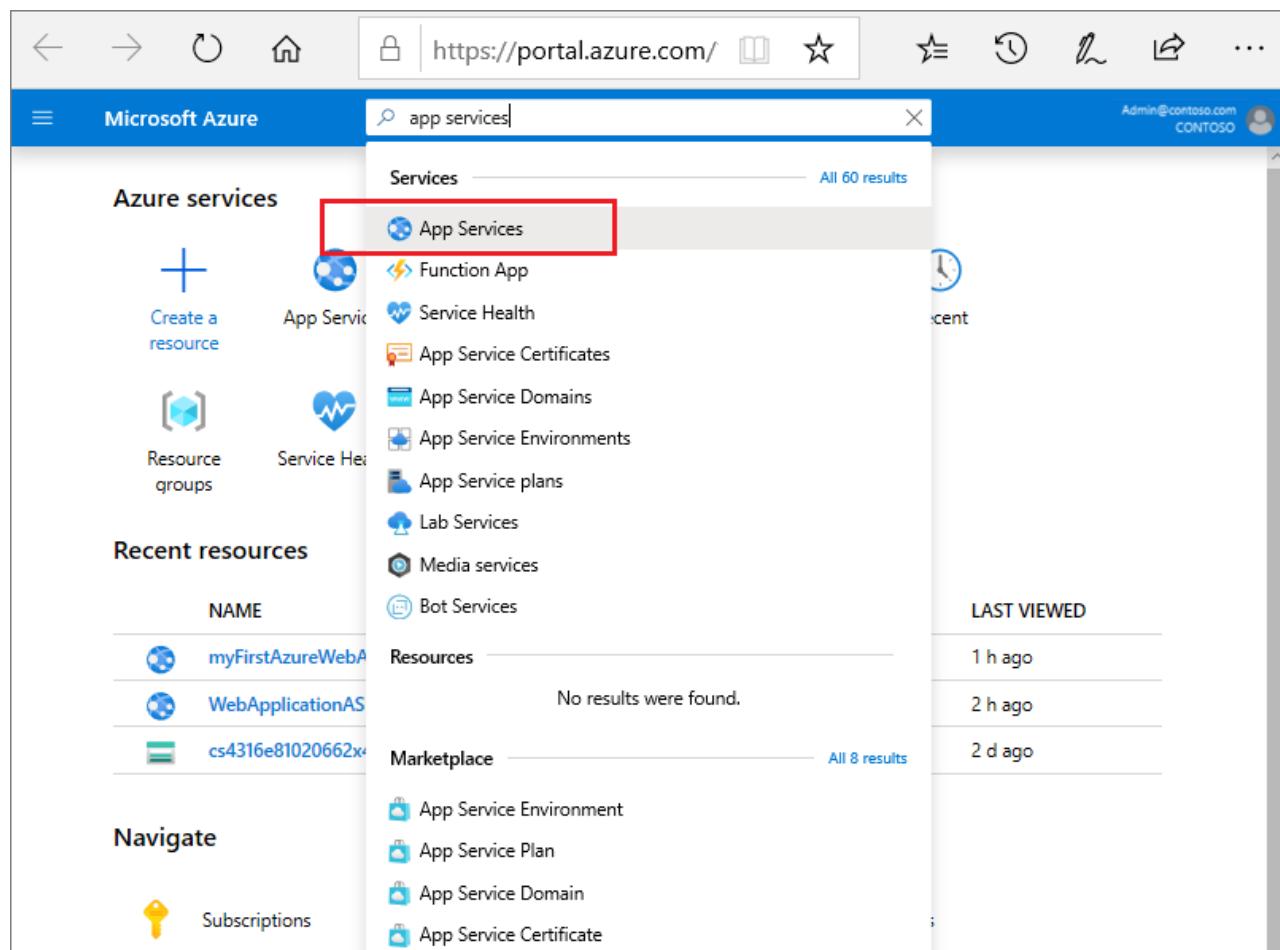
2/25/2020 • 8 minutes to read • [Edit Online](#)

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" (Create a resource, Resource groups), "Recent resources" (myFirstAzureWebA, WebApplicationAS, cs4316e81020662x), and "Navigate" (Subscriptions). The main content area is titled "Services" and shows a list of items under "All 60 results". One item, "App Services", is highlighted with a red box. Other items include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, Bot Services, Resources (which shows "No results were found."), and Marketplace (which shows "All 8 results"). On the right side of the main content area, there are several small icons representing different Azure services like Storage, Container Registry, and App Service Environment.

In the app's left menu, select **Configuration > Application settings**.

The screenshot shows the 'my-core-app - Configuration' blade in the Azure portal. On the left, a navigation menu lists 'Security', 'Deployment' (with 'Quickstart', 'Deployment slots', and 'Deployment Center'), 'Settings' (with 'Configuration' highlighted), and other options like 'Application settings (Classic)', 'Authentication / Authorization', 'Application Insights', 'Identity', 'Backups', 'Custom domains', 'SSL settings', 'Networking', 'Scale up (App Service plan)', and 'Scale out (App Service plan)'. The main content area has tabs for 'Application settings' (selected), 'General settings', 'Default documents', and 'Path mappings'. The 'Application settings' tab displays a table with columns 'Name', 'Value', and 'deployment...'. A note says '(no application settings to display)'. Below it, the 'Connection strings' section also shows a table with columns 'Name', 'Value', 'Type', and 'deployment...'. A note says '(no connection strings to display)'. At the top right, there are 'Save' and 'Discard' buttons.

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 sidebar on the left has a 'Configuration' section highlighted with a red box. The main area has tabs for 'Application settings', 'General settings', 'Default documents', and 'Path mappings', with 'Application settings' being the active tab. Below the tabs, there's a section titled 'Application settings' with a note about encryption and environment variables. A table lists application settings, showing one entry: Name (no application settings to display). Below this is a section for 'Connection strings' with a note about encryption. A table lists connection strings, showing one entry: Name (no connection strings to display). At the bottom of the blade are buttons for 'New application setting', 'Show values', 'Advanced edit', and 'Filter'.

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 screenshot shows the Azure portal configuration interface for an app named "my-core-app". The left sidebar has a tree view with "Configuration" selected. The main content area has tabs for "Application settings", "General settings", "Default documents" (which is highlighted with a red box), and "Path mappings". Below these tabs, there are two sections: "Application settings" and "Connection strings", each with a "New [type] setting" button and a table for managing values. Both sections currently show "(no [type] settings to display)".

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 configuration interface for an app service named 'my-core-app'. The left sidebar lists various settings like Security, Deployment, and Settings, with 'Configuration' selected and highlighted with a red box. The main content area has tabs for Application settings, General settings, Default documents, and Path mappings, with 'Path mappings' also highlighted with a red box. Below these tabs, there are sections for Application settings and Connection strings, each with their own sub-sections and buttons for adding new items or editing existing ones.

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 `handler.cgi`.
- **Script processor.** The absolute path of the script processor to you. Requests to files that match the file extension are processed by the script processor. Use the path `D:\home\site\wwwroot` to refer to your app's root directory.
- **Arguments.** Optional command-line arguments for the script processor.

Each app has the default root path (`/`) mapped to `D:\home\site\wwwroot`, where your code is deployed by default. If your app root is in a different folder, or if your repository has more than one application, you can edit or add virtual applications and directories here. 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 PHP in Azure App Service

2/4/2020 • 5 minutes to read • [Edit Online](#)

Introduction

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](#), provide a custom PHP runtime, and enable extensions. To use App Service, sign up for the [free trial](#). To get the most from this guide, you should first create a PHP app in App Service.

How to: Change the built-in PHP version

By default, PHP 5.6 is installed and immediately available for use when you create an App Service app. The best way to see the available release revision, its default configuration, and the enabled extensions is to deploy a script that calls the [phpinfo\(\)](#) function.

PHP 7.0 and PHP 7.2 versions are also available, but not enabled by default. To update the PHP version, follow one of these methods:

Azure portal

1. Browse to your app in the [Azure portal](#) and scroll to the **Configuration** page.
2. From **Configuration**, select **General Settings** and choose the new PHP version.
3. Click the **Save** button at the top of the **General settings** blade.

Azure CLI

To use the Azure Command-Line Interface, you must [Install the Azure CLI](#) on your computer.

1. Open Terminal, and login to your account.

```
az login
```

2. Check to see the list of supported runtimes.

```
az webapp list-runtimes | grep php
```

3. Set the PHP version for the app.

```
az webapp config set --php-version {5.6 | 7.0 | 7.1 | 7.2} --name {app-name} --resource-group {resource-group-name}
```

4. The PHP version is now set. You can confirm these settings:

```
az webapp show --name {app-name} --resource-group {resource-group-name}
```

How to: Change the built-in PHP configurations

For any built-in PHP runtime, you can change any of the configuration options by following these steps. (For information about `php.ini` directives, see [List of `php.ini` directives](#).)

Changing PHP_INI_USER, PHP_INI_PERDIR, PHP_INI_ALL configuration settings

1. Add a `.user.ini` file to your root directory.
2. Add configuration settings to the `.user.ini` file using the same syntax you would use in a `php.ini` file. For example, if you wanted to turn on the `display_errors` setting and set `upload_max_filesize` setting to 10M, your `.user.ini` file would contain this text:

```
; Example Settings
display_errors=On
upload_max_filesize=10M

; OPTIONAL: Turn this on to write errors to d:\home\LogFiles\php_errors.log
; log_errors=On
```

3. Deploy your app.
4. Restart the app. (Restarting is necessary because the frequency with which PHP reads `.user.ini` files is governed by the `user_ini.cache_ttl` setting, which is a system level setting and is 300 seconds (5 minutes) by default. Restarting the app forces PHP to read the new settings in the `.user.ini` file.)

As an alternative to using a `.user.ini` file, you can use the `ini_set()` function in scripts to set configuration options that are not system-level directives.

Changing PHP_INI_SYSTEM configuration settings

1. Add an App Setting to your app with the key `PHP_INI_SCAN_DIR` and value `d:\home\site\ini`
2. Create an `settings.ini` file using Kudu Console (<http://<site-name>.scm.azurewebsite.net>) in the `d:\home\site\ini` directory.
3. Add configuration settings to the `settings.ini` file using the same syntax you would use in a `php.ini` file. For example, if you wanted to point the `curl.cainfo` setting to a `*.crt` file and set 'wincache.maxfilesize' setting to 512K, your `settings.ini` file would contain this text:

```
; Example Settings
curl.cainfo="%ProgramFiles(x86)%\Git\bin\curl-ca-bundle.crt"
wincache.maxfilesize=512
```

4. To reload the changes, restart your app.

How to: Enable extensions in the default PHP runtime

As noted in the previous section, the best way to see the default PHP version, its default configuration, and the enabled extensions is to deploy a script that calls `phpinfo()`. To enable additional extensions, by following these steps:

Configure via ini settings

1. Add a `ext` directory to the `d:\home\site` directory.
2. Put `.dll` extension files in the `ext` directory (for example, `php_xdebug.dll`). Make sure that the extensions are compatible with default version of PHP and are VC9 and non-thread-safe (nts) compatible.
3. Add an App Setting to your app with the key `PHP_INI_SCAN_DIR` and value `d:\home\site\ini`
4. Create an `ini` file in `d:\home\site\ini` called `extensions.ini`.
5. Add configuration settings to the `extensions.ini` file using the same syntax you would use in a `php.ini`

file. For example, if you wanted to enable the MongoDB and XDebug extensions, your `extensions.ini` file would contain this text:

```
; Enable Extensions  
extension=d:\home\site\ext\php_mongo.dll  
zend_extension=d:\home\site\ext\php_xdebug.dll
```

6. Restart your app to load the changes.

Configure via App Setting

1. Add a `bin` directory to the root directory.
2. Put `.dll` extension files in the `bin` directory (for example, `php_xdebug.dll`). Make sure that the extensions are compatible with default version of PHP and are VC9 and non-thread-safe (nts) compatible.
3. Deploy your app.
4. Browse to your app in the Azure portal and click on the **Configuration** located below **Settings** section.
5. From the **Configuration** blade, select **Application Settings**.
6. In the **Application settings** section, click on **+ New application setting** and create a **PHP_EXTENSIONS** key. The value for this key would be a path relative to website root: `bin\your-ext-file`.
7. Click the **Update** button at the bottom then click **Save** above the **Application settings** tab.

Zend extensions are also supported by using a **PHP_ZENDEXTENSIONS** key. To enable multiple extensions, include a comma-separated list of `.dll` files for the app setting value.

How to: Use a custom PHP runtime

Instead of the default PHP runtime, App Service can use a PHP runtime that you provide to execute PHP scripts. The runtime that you provide can be configured by a `php.ini` file that you also provide. To use a custom PHP runtime with App Service, following these steps.

1. Obtain a non-thread-safe, VC9 or VC11 compatible version of PHP for Windows. Recent releases of PHP for Windows can be found here: <https://windows.php.net/download/>. Older releases can be found in the archive here: <https://windows.php.net/downloads/releases/archives/>.
2. Modify the `php.ini` file for your runtime. Any configuration settings that are system-level-only directives are ignored by App Service. (For information about system-level-only directives, see [List of php.ini directives](#)).
3. Optionally, add extensions to your PHP runtime and enable them in the `php.ini` file.
4. Add a `bin` directory to your root directory, and put the directory that contains your PHP runtime in it (for example, `bin\php`).
5. Deploy your app.
6. Browse to your app in the Azure portal and click on the **Configuration** blade.
7. From the **Configuration** blade, select **Path mappings**.
8. Click **+ New Handler** and add `*.php` to the Extension field and add the path to the `php-cgi.exe` executable in **Script processor**. If you put your PHP runtime in the `bin` directory in the root of your application, the path is `D:\home\site\wwwroot\bin\php\php-cgi.exe`.
9. At the bottom, click **Update** to finish adding the handler mapping.
10. Click **Save** to save changes.

How to: Enable Composer automation in Azure

By default, App Service doesn't do anything with `composer.json`, if you have one in your PHP project. If you use [Git deployment](#), you can enable `composer.json` processing during `git push` by enabling the Composer extension.

NOTE

You can [vote for first-class Composer support in App Service here!](#)

1. In your PHP app's blade in the [Azure portal](#), click **Tools > Extensions**.

The screenshot shows the Azure portal interface. On the left, the 'cephalin-php' web app blade is open, displaying basic app details like resource group, status, location, and subscription information. The 'Tools' button in the top navigation bar is highlighted with a red box. On the right, the 'Tools' pane is open, listing various tools: Mitigate, Live HTTP traffic, Clone App, Zend Z-Ray, Performance test, Console, Visual Studio Online, Kudu, and Extensions. The 'Extensions' item in the list is also highlighted with a red box.

2. Click **Add**, then click **Composer**.

The screenshot shows three windows. On the left is the 'Installed extensions' blade for the 'cephalin-php' app, with an 'Add' button highlighted by a red box. In the center is the 'Add extension' wizard, step 1: 'Choose Extension'. Step 2: 'Legal Terms' is shown below it. On the right is the 'Choose extension' list, showing various options like Azure Web Site Logs Browser, Site Replicator, New Relic, phpMyAdmin, File Counter (sample), Composer, Php Manager, and Azure Image Optimizer. The 'Composer' option is highlighted with a red box.

3. Click **OK** to accept legal terms. Click **OK** again to add the extension.

The **Installed extensions** blade shows the Composer extension.

4. Now, in a terminal window on your local machine, perform `git add`, `git commit`, and `git push` to your app. Notice that Composer is installing dependencies defined in `composer.json`.

```
PS D:\php-get-started> git push azure master
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 336 bytes | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Updating branch 'master'.
remote: Updating submodules.
remote: Preparing deployment for commit id '997332926e'.
remote: Running custom deployment command...
remote: Running deployment command...
remote: Install Dependencies with Composer
remote: Loading composer repositories with package information
remote: Installing dependencies
remote: .....
remote: Nothing to install or update
remote: Writing lock file
remote: Generating optimized autoload files
remote: Handling Basic Web Site deployment
```

Next steps

For more information, see the [PHP Developer Center](#).

Configure a Windows Java app for Azure App Service

2/18/2020 • 12 minutes to read • [Edit Online](#)

Azure App Service lets Java developers quickly build, deploy, and scale their Tomcat web applications on a fully managed Windows-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 using App Service. If you've never used Azure App Service, you should read through the [Java quickstart](#) first. General questions about using App Service that aren't specific to the Java development are answered in the [App Service Windows FAQ](#).

Deploying your app

You can use [Azure Web App Plugin for Maven](#) to deploy your .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 to Java SE, use the `/api/zipdeploy/` endpoint of the Kudu site. For more information on this API, please see [this documentation](#).

Do not deploy your .war using FTP. The FTP tool is designed to upload startup scripts, dependencies, or other runtime files. It is not the optimal choice for deploying web apps.

Logging and debugging apps

Performance reports, traffic visualizations, and health checkups are available for each app through the Azure portal. For more information, see [Azure App Service diagnostics overview](#).

Stream diagnostic logs

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 `/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](#).

Customization and tuning

Azure App Service 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, create an [app setting](#) named `JAVA_OPTS` with the options. App Service passes this setting as an environment variable to the Java runtime when it starts.

In the Azure portal, under **Application Settings** for the web app, create a new app setting named `JAVA_OPTS` 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.

Turn on web sockets

Turn on support for web sockets in the Azure portal in the **Application settings** for the application. You'll need to restart the application for the setting to take effect.

Turn on web socket support using the Azure CLI with the following command:

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --web-sockets-enabled true
```

Then restart your application:

```
az webapp stop --name <app-name> --resource-group <resource-group-name>
az webapp start --name <app-name> --resource-group <resource-group-name>
```

Set default character encoding

In the Azure portal, under **Application Settings** for the web app, create a new app setting named `JAVA_OPTS` with value `-Dfile.encoding=UTF-8`.

Alternatively, you can configure the app setting using the App Service Maven plugin. Add the setting name and value tags in the plugin configuration:

```
<appSettings>
  <property>
    <name>JAVA_OPTS</name>
    <value>-Dfile.encoding=UTF-8</value>
  </property>
</appSettings>
```

Pre-Compile JSP files

To improve performance of Tomcat applications, you can compile your JSP files before deploying to App Service. You can use the [Maven plugin](#) provided by Apache Sling, or using this [Ant build file](#).

Secure applications

Java applications running in App Service have the same set of [security best practices](#) as other applications.

Authenticate users (Easy Auth)

Set up app authentication in the Azure portal with the **Authentication and Authorization** option. From there, you can enable authentication using Azure Active Directory or social logins like Facebook, Google, or GitHub. Azure portal configuration only works when configuring a single authentication provider. For more information, see [Configure your App Service app to use Azure Active Directory login](#) and the related articles for other identity providers. If you need to enable multiple sign-in providers, follow the instructions in the [customize 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`.

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](#).

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 New Relic 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. Use the [Kudu console](#) to create a new directory `/home/site/wwwroot/apm`.
5. Upload the unpacked New Relic 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. Use the [Kudu console](#) to create a new directory `/home/site/wwwroot/apm`.
4. Upload the Java agent files into a directory under `/home/site/wwwroot/apm`. The files for your agent should be in `/home/site/wwwroot/apm/appdynamics`.
5. In the Azure portal, browse to your application in App Service and create a new Application Setting.
 - 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.

If you already have an environment variable for `JAVA_OPTS` or `CATALINA_OPTS`, append the `-javaagent:/...` option to the end of the current value.

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>	Download
MySQL	<code>com.mysql.jdbc.Driver</code>	Download (Select "Platform Independent")
SQL Server	<code>com.microsoft.sqlserver.jdbc.SQLServerDriver</code>	Download

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>
```

Finalize configuration

Finally, we will place the driver JARs in the Tomcat classpath and restart your App Service. Ensure that the JDBC driver files are available to the Tomcat classloader by placing them in the `/home/tomcat/lib` directory. (Create this directory if it does not already exist.) To upload these files to your App Service instance, perform the following steps:

1. In the [Cloud Shell](#), install the webapp extension:

```
az extension add --name webapp
```

2. Run the following CLI command to create an SSH tunnel from your local system to App Service:

```
az webapp remote-connection create --resource-group <resource-group-name> --name <app-name> --port <port-on-local-machine>
```

3. Connect to the local tunneling port with your SFTP client and upload the files to the `/home/tomcat/lib` folder.

Alternatively, you can use an FTP client to upload the JDBC driver. Follow these[instructions for getting your FTP credentials](#).

Configuring Tomcat

To edit Tomcat's `server.xml` or other configuration files, first take a note of your Tomcat major version in the portal.

1. Find the Tomcat home directory for your version by running the `env` command. Search for the environment variable that begins with `AZURE_TOMCAT` and matches your major version. For example, `AZURE_TOMCAT85_HOME`

points to the Tomcat directory for Tomcat 8.5.

2. Once you have identified the Tomcat home directory for your version, copy the configuration directory to `D:\home`. For example, if `AZURE_TOMCAT85_HOME` had a value of `D:\Program Files (x86)\apache-tomcat-8.5.37`, the new path of the copied directory would be `D:\home\apache-tomcat-8.5.37`.

Finally, restart your App Service. Your deployments should go to `D:\home\site\wwwroot\webapps` just as before.

Configure Java SE

When running a JAR application on Java SE on Windows, `server.port` is passed as a command line option as your application starts. You can manually resolve the HTTP port from the environment variable, `HTTP_PLATFORM_PORT`. The value of this environment variable will be the HTTP port your application should listen on.

Java runtime statement of support

JDK versions and maintenance

Azure's supported Java Development Kit (JDK) is [Zulu](#) provided through [Azul Systems](#).

Major version updates will be provided through new runtime options in Azure App Service for Windows. Customers update to these newer versions of Java by configuring their App Service deployment and are responsible for testing and ensuring the major update meets their needs.

Supported JDKs are automatically patched on a quarterly basis in January, April, July, and October of each year. For more information on Java on Azure, please see [this support document](#).

Security updates

Patches and fixes for major security vulnerabilities will be released as soon as they become available from Azul Systems. A "major" vulnerability is defined by a base score of 9.0 or higher on the [NIST Common Vulnerability Scoring System, version 2](#).

Tomcat 8.0 has reached [End of Life \(EOL\) as of September 30, 2018](#). While the runtime is still available on Azure App Service, Azure will not apply security updates to Tomcat 8.0. If possible, migrate your applications to Tomcat 8.5 or 9.0. Both Tomcat 8.5 and 9.0 are available on Azure App Service. See the [official Tomcat site](#) for more information.

Deprecation and retirement

If a supported Java runtime will be retired, Azure developers using the affected runtime will be given a deprecation notice at least six months before the runtime is retired.

Local development

Developers can download the Production Edition of Azul Zulu Enterprise JDK for local development from [Azul's download site](#).

Development support

Product support for the [Azure-supported Azul Zulu JDK](#) is available through Microsoft when developing for Azure or [Azure Stack](#) with a [qualified Azure support plan](#).

Runtime support

Developers can [open an issue](#) with the Azul Zulu JDKs through Azure Support if they have a [qualified support plan](#).

Next steps

This topic provides the Java Runtime statement of support for Azure App Service on Windows.

- To learn more about hosting web applications with Azure App Service see [App Service overview](#).
- For information about Java on Azure development see [Azure for Java Dev Center](#).

Integrate your app with an Azure Virtual Network

2/26/2020 • 23 minutes to read • [Edit Online](#)

This document describes the Azure App Service virtual network integration feature and how to set it up with apps in the [Azure App Service](#). [Azure Virtual Networks](#) (VNets) allow you to place many of your Azure resources in a non-internet routable network.

The Azure App Service has two variations.

1. The multi-tenant systems that support the full range of pricing plans except Isolated
2. The App Service Environment (ASE), which deploys into your VNet and supports Isolated pricing plan apps

This document goes through the VNet Integration feature, which is for use in the multi-tenant App Service. If your app is in [App Service Environment](#), then it's already in a VNet and doesn't require use of the VNet Integration feature to reach resources in the same VNet. For details on all of the App Service networking features, read [App Service networking features](#)

VNet Integration gives your web app access to resources in your virtual network but doesn't grant inbound private access to your web app from the VNet. Private site access refers to making your app only accessible from a private network such as from within an Azure virtual network. VNet Integration is only for making outbound calls from your app into your VNet. The VNet Integration feature behaves differently when used with VNets in the same region and with VNets in other regions. The VNet Integration feature has two variations.

1. Regional VNet Integration - When connecting to Resource Manager VNets in the same region, you must have a dedicated subnet in the VNet you are integrating with.
2. Gateway required VNet Integration - When connecting to VNets in other regions or to a Classic VNet in the same region you need a Virtual Network gateway provisioned in the target VNet.

The VNet Integration features:

- require a Standard, Premium, PremiumV2, or Elastic Premium pricing plan
- support TCP and UDP
- work with App Service apps, and Function apps

There are some things that VNet Integration doesn't support including:

- mounting a drive
- AD integration
- NetBios

Gateway required VNet Integration only provides access to resources in the target VNet or in networks connected to the target VNet with peering or VPNs. Gateway required VNet Integration doesn't enable access to resources available across ExpressRoute connections or works with service endpoints.

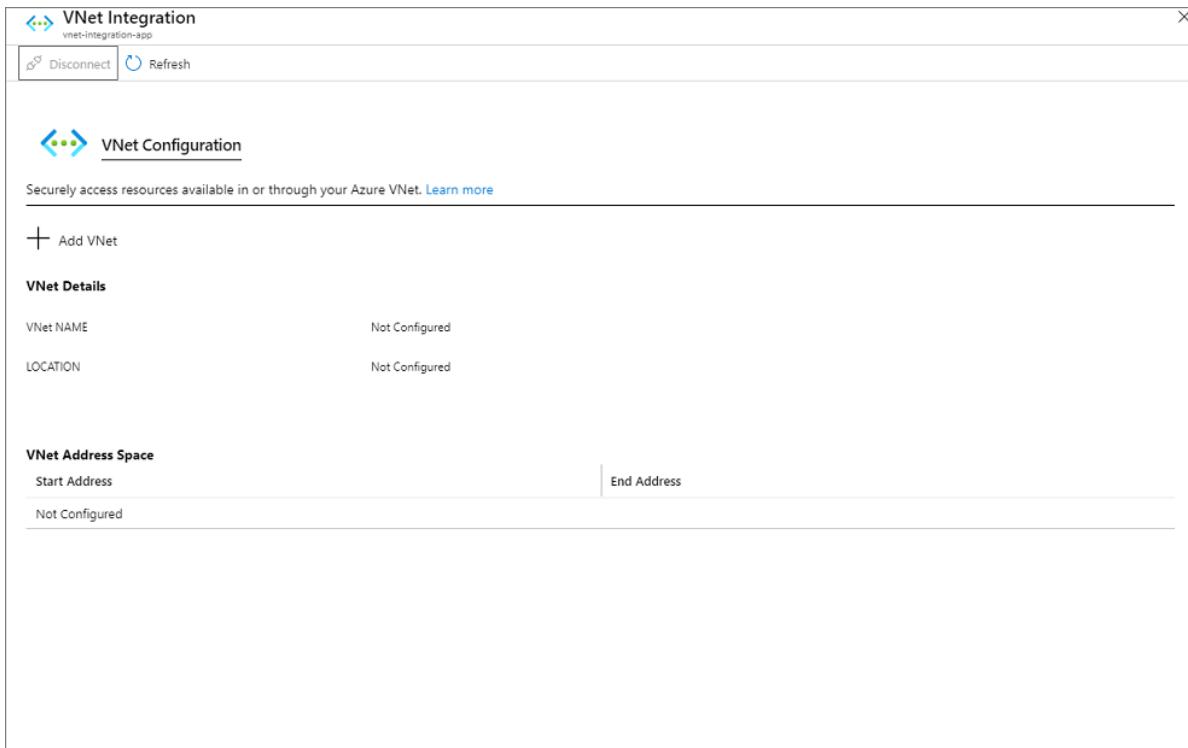
Regardless of the version used, VNet Integration gives your web app access to resources in your virtual network but doesn't grant inbound private access to your web app from the virtual network. Private site access refers to making your app only accessible from a private network such as from within an Azure virtual network. VNet Integration is only for making outbound calls from your app into your VNet.

Enable VNet Integration

1. Go to the Networking UI in the App Service portal. Under VNet Integration, select "[Click here to](#)

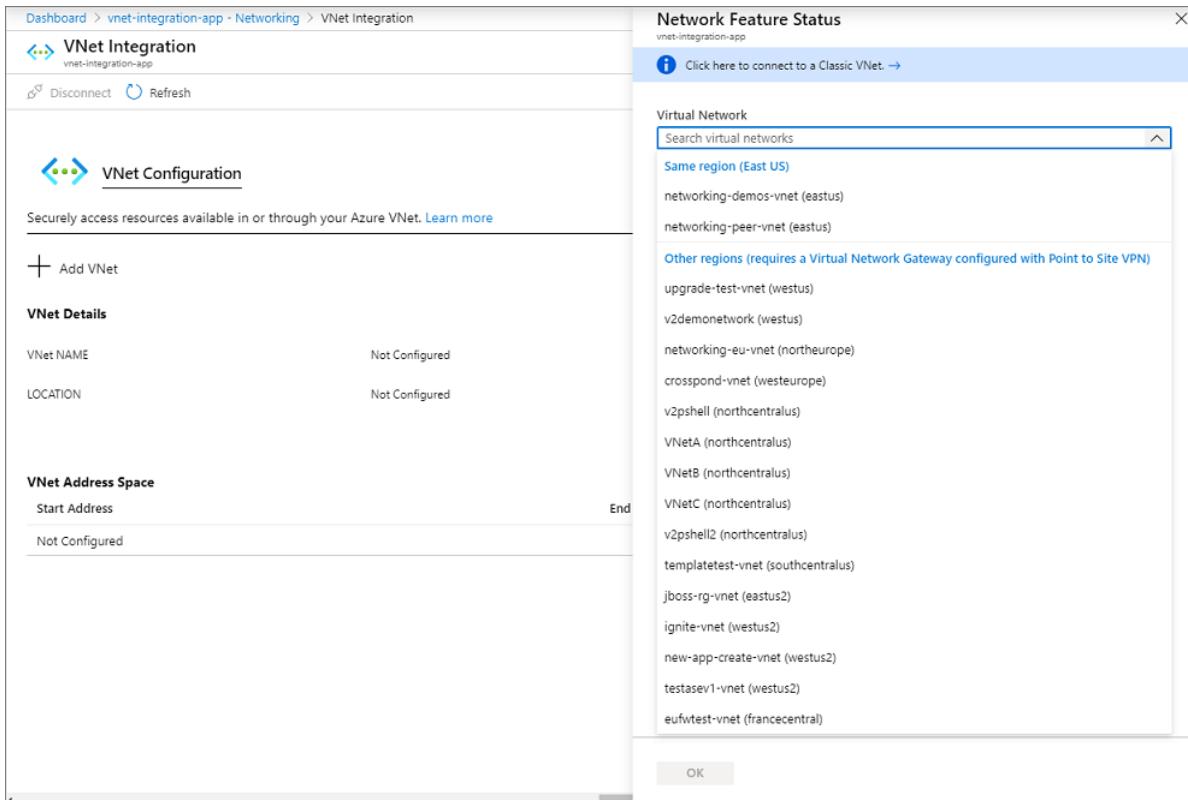
configure".

2. Select **Add VNet**.



The screenshot shows the 'VNet Configuration' page. At the top, there's a header with a 'Disconnect' and 'Refresh' button. Below the header, a section titled 'VNet Configuration' with a subtitle 'Securely access resources available in or through your Azure VNet. [Learn more](#)' is visible. A large 'Add VNet' button is prominently displayed. The page is divided into sections: 'VNet Details' (VNet NAME: Not Configured, LOCATION: Not Configured), 'VNet Address Space' (Start Address: Not Configured, End Address: Not Configured), and other configuration options.

3. The drop-down list will contain all of the Resource Manager VNets in your subscription in the same region and below that is a list of all of the Resource Manager VNets in all other regions. Select the VNet you wish to integrate with.



The screenshot shows the 'VNet Configuration' page with a modal window titled 'Network Feature Status' overlaid. The modal contains a message 'Click here to connect to a Classic VNet. →'. Below this, there are two sections: 'Virtual Network' and 'Other regions (requires a Virtual Network Gateway configured with Point to Site VPN)'. The 'Virtual Network' section shows a search bar and a list of VNets in the 'Same region (East US)': networking-demos-vnet (eastus) and networking-peer-vnet (eastus). The 'Other regions' section lists VNets from various regions: upgrade-test-vnet (westus), v2demonetwerk (westus), networking-eu-vnet (northeurope), crosspond-vnet (westeurope), v2pshell (northcentralus), VNetA (northcentralus), VNetB (northcentralus), VNetC (northcentralus), v2pshell2 (northcentralus), templatedtest-vnet (southcentralus), jboss-rg-vnet (eastus2), ignite-vnet (westus2), new-app-create-vnet (westus2), testasev1-vnet (westus2), and eufwtest-vnet (francecentral). An 'OK' button is at the bottom of the modal.

- If the VNet is in the same region, then either create a new subnet or pick an empty pre-existing subnet.
- To select a VNet in another region, you must have a Virtual Network gateway provisioned with point to site enabled.

- To integrate with a Classic VNet, instead of clicking the VNet drop down, select **Click here to connect to a Classic VNet**. Select the desired Classic VNet. The target VNet must already have a Virtual Network gateway provisioned with point to site enabled.

During the integration, your app is restarted. When integration is completed, you will see details on the VNet you are integrated with.

Once your app is integrated with your VNet, it will use the same DNS server that your VNet is configured with, unless it is Azure DNS Private Zones. You currently cannot use VNet Integration with Azure DNS Private Zones.

Regional VNet Integration

Using regional VNet Integration enables your app to access:

- resources in the VNet in the same region that you integrate with
- resources in VNets peered to your VNet that are in the same region
- service endpoint secured services
- resources across ExpressRoute connections
- resources in the VNet you are connected to
- resources across peered connections including ExpressRoute connections
- private endpoints

When using VNet Integration with VNets in the same region, you can use the following Azure Networking features:

- Network Security Groups(NSGs) - You can block outbound traffic with a Network Security Group that is placed on your integration subnet. The inbound rules do not apply as you cannot use VNet Integration to provide inbound access to your web app.
- Route Tables (UDRs) - You can place a route table on the integration subnet to send outbound traffic where you want.

By default, your app will only route RFC1918 traffic into your VNet. If you want to route all of your outbound traffic into your VNet, apply the app setting WEBSITE_VNET_ROUTE_ALL to your app. To configure the app setting:

1. Go to the Configuration UI in your app portal. Select **New application setting**
2. Type **WEBSITE_VNET_ROUTE_ALL** in the Name field and **1** in the Value field

The screenshot shows the Azure portal's configuration interface for an App Service application named 'vnet-integration-app'. On the left, there's a navigation sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings (Configuration, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs), and a search bar at the top.

In the main content area, a modal window titled 'Add/Edit application setting' is open. It contains a table with one row. The 'Name' column has the value 'WEBSITE_VNET_ROUTE_ALL' and the 'Value' column has the value '1'. There is also a checkbox labeled 'Deployment slot setting' which is currently unchecked. At the bottom of the modal are two buttons: 'OK' and 'Cancel'.

3. Select **OK**

4. Select **Save**

If you route all of your outbound traffic into your VNet, then it will be subject to the NSGs and UDRs that are applied to your integration subnet. When you route all of your outbound traffic into your VNet, your outbound addresses will still be the outbound addresses that are listed in your app properties unless you provide routes to send the traffic elsewhere.

There are some limitations with using VNet Integration with VNets in the same region:

- You cannot reach resources across global peering connections
- The feature is only available from newer App Service scale units that support PremiumV2 App Service plans.
- The integration subnet can only be used by only one App Service plan
- The feature cannot be used by Isolated plan apps that are in an App Service Environment
- The feature requires an unused subnet that is a /27 with 32 addresses or larger in a Resource Manager VNet
- The app and the VNet must be in the same region
- You cannot delete a VNet with an integrated app. Remove the integration before deleting the VNet.
- You can only integrate with VNets in the same subscription as the web app
- You can have only one regional VNet Integration per App Service plan. Multiple apps in the same App Service plan can use the same VNet.
- You cannot change the subscription of an app or an App Service plan while there is an app that is using Regional VNet Integration

One address is used for each App Service plan instance. If you scale your app to five instances, then five addresses are used. Since subnet size cannot be changed after assignment, you must use a subnet that is large enough to accommodate whatever scale your app may reach. A /26 with 64 addresses is the recommended size. A /26 with 64 addresses will accommodate a Premium App Service plan with 30 instances. When you scale an App Service plan up or down, you need twice as many addresses for a short period of time.

If you want your apps in another App Service plan to reach a VNet that is connected to already by apps in another App Service plan, you need to select a different subnet than the one being used by the pre-existing VNet

Integration.

The feature is in preview for Linux. The Linux form of the feature only supports making calls to RFC 1918 addresses (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16).

Web App for Containers

If you use App Service on Linux with the built-in images, regional VNet Integration works without additional changes. If you use Web App for Containers, you need to modify your docker image in order to use VNet Integration. In your docker image, use the PORT environment variable as the main web server's listening port, instead of using a hardcoded port number. The PORT environment variable is automatically set by App Service platform at the container startup time. If you are using SSH, then the SSH daemon must be configured to listen on the port number specified by the SSH_PORT environment variable when using regional VNet integration. There is no support for gateway required VNet Integration on Linux.

Service Endpoints

Regional VNet Integration enables you to use service endpoints. To use service endpoints with your app, use regional VNet Integration to connect to a selected VNet and then configure service endpoints on the subnet you used for the integration.

Network Security Groups

Network Security Groups enable you to block inbound and outbound traffic to resources in a VNet. A web app using regional VNet Integration can use [Network Security Group](#) to block outbound traffic to resources in your VNet or the internet. To block traffic to public addresses, you must have the application setting WEBSITE_VNET_ROUTE_ALL set to 1. The inbound rules in an NSG do not apply to your app as VNet Integration only affects outbound traffic from your app. To control inbound traffic to your web app, use the Access Restrictions feature. An NSG that is applied to your integration subnet will be in effect regardless of any routes applied to your integration subnet. If WEBSITE_VNET_ROUTE_ALL was set to 1 and you did not have any routes affecting public address traffic on your integration subnet, all of your outbound traffic would still be subject to NSGs assigned to your integration subnet. If WEBSITE_VNET_ROUTE_ALL was not set, NSGs would only be applied to RFC1918 traffic.

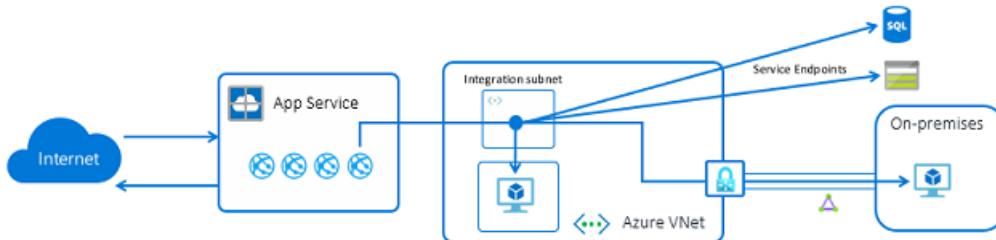
Routes

Route Tables enable you to route outbound traffic from your app to wherever you want. By default, route tables will only affect your RFC1918 destination traffic. If you set WEBSITE_VNET_ROUTE_ALL to 1, then all of your outbound calls will be affected. Routes that are set on your integration subnet will not affect replies to inbound app requests. Common destinations can include firewall devices or gateways. If you want to route all outbound traffic on-premises, you can use a route table to send all outbound traffic to your ExpressRoute gateway. If you do route traffic to a gateway, be sure to set routes in the external network to send any replies back.

Border Gateway Protocol (BGP) routes will also affect your app traffic. If you have BGP routes from something like an ExpressRoute gateway, your app outbound traffic will be affected. By default, BGP routes will only affect your RFC1918 destination traffic. If WEBSITE_VNET_ROUTE_ALL is set to 1, then all outbound traffic can be affected by your BGP routes.

How Regional VNet Integration works

Apps in the App Service are hosted on worker roles. The Basic and higher pricing plans are dedicated hosting plans where there are no other customers workloads running on the same workers. Regional VNet Integration works by mounting virtual interfaces with addresses in the delegated subnet. Because the from address is in your VNet, it can access to most things in or through your VNet just like a VM in your VNet would. The networking implementation is different than running a VM in your VNet and that is why some networking features are not yet available while using this feature.



When regional VNet Integration is enabled, your app will still make outbound calls to the internet through the same channels as normal. The outbound addresses that are listed in the app properties portal are still the addresses used by your app. What changes for your app are, calls to service endpoint secured services or RFC 1918 addresses goes into your VNet. If WEBSITE_VNET_ROUTE_ALL is set to 1, then all outbound traffic can be sent into your VNet.

The feature only supports one virtual interface per worker. One virtual interface per worker means one regional VNet Integration per App Service plan. All of the apps in the same App Service plan can use the same VNet Integration but if you need an app to connect to an additional VNet, you will need to create another App Service plan. The virtual interface used is not a resource that customers have direct access to.

Due to the nature of how this technology operates, the traffic that is used with VNet Integration does not show up in Network Watcher or NSG flow logs.

Gateway required VNet Integration

Gateway required VNet Integration supports connecting to a VNet in another region, or to a Classic VNet.

Gateway required VNet Integration:

- Enables an app to connect to only 1 VNet at a time
- Enables up to five VNets to be integrated within an App Service Plan
- Allows the same VNet to be used by multiple apps in an App Service Plan without impacting the total number that can be used by an App Service plan. If you have six apps using the same VNet in the same App Service plan, that counts as 1 VNet being used.
- Supports a 99.9% SLA due to the SLA on the gateway
- Enables your apps to use the DNS that the VNet is configured with
- Requires a Virtual Network route-based gateway configured with SSTP point-to-site VPN before it can be connected to app.

You can't use gateway required VNet Integration:

- With Linux apps
- With a VNet connected with ExpressRoute
- To access Service Endpoints secured resources
- With a coexistence gateway that supports both ExpressRoute and point to site/site to site VPNs

Set up a gateway in your VNet

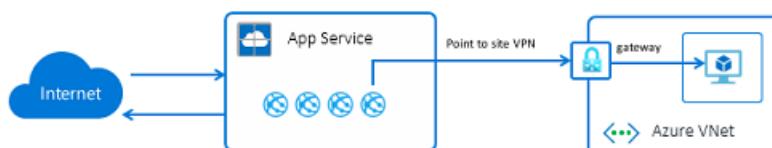
To create a gateway:

1. [Create a gateway subnet](#) in your VNet.
2. [Create the VPN gateway](#). Select a route-based VPN type.
3. [Set the point to site addresses](#). If the gateway isn't in the basic SKU, then IKEV2 must be disabled in the point to site configuration and SSTP must be selected. The point to site address space must be in the RFC 1918 address blocks, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16

If you are just creating the gateway for use with App Service VNet Integration, then you do not need to upload a certificate. Creating the gateway can take 30 minutes. You will not be able to integrate your app with your VNet until the gateway is provisioned.

How gateway required VNet Integration works

Gateway required VNet Integration built on top of point to site VPN technology. Point to site VPNs limits network access to just the virtual machine hosting the app. Apps are restricted to only send traffic out to the internet, through Hybrid Connections or through VNet Integration. When your app is configured with the portal to use gateway required VNet Integration, a complex negotiation is managed on your behalf to create and assign certificates on the gateway and the application side. The end result is that the workers used to host your apps are able to directly connect to the virtual network gateway in the selected VNet.



Accessing on-premises resources

Apps can access on-premises resources by integrating with VNets that have site-to-site connections. If you are using the gateway required VNet Integration, you need to update your on-premises VPN gateway routes with your point-to-site address blocks. When the site-to-site VPN is first set up, the scripts used to configure it should set up routes properly. If you add the point-to-site addresses after you create your site-to-site VPN, you need to update the routes manually. Details on how to do that vary per gateway and are not described here. You cannot have BGP configured with a site-to-site VPN connection.

There is no additional configuration required for the regional VNet Integration feature to reach through your VNet, and to on-premises. You simply need to connect your VNet to on-premises using ExpressRoute or a site-to-site VPN.

NOTE

The gateway required VNet Integration feature doesn't integrate an app with a VNet that has an ExpressRoute Gateway. Even if the ExpressRoute Gateway is configured in [coexistence mode](#) the VNet Integration doesn't work. If you need to access resources through an ExpressRoute connection, then you can use the regional VNet Integration feature or an [App Service Environment](#), which runs in your VNet.

Peering

If you are using peering with the regional VNet Integration, you do not need to do any additional configuration.

If you are using the gateway required VNet Integration with peering, you need to configure a few additional items. To configure peering to work with your app:

1. Add a peering connection on the VNet your app connects to. When adding the peering connection, enable **Allow virtual network access** and check **Allow forwarded traffic** and **Allow gateway transit**.
2. Add a peering connection on the VNet that is being peered to the VNet you are connected to. When adding the peering connection on the destination VNet, enable **Allow virtual network access** and check **Allow forwarded traffic** and **Allow remote gateways**.
3. Go to the App Service plan > Networking > VNet Integration UI in the portal. Select the VNet your app connects to. Under the routing section, add the address range of the VNet that is peered with the VNet your app is connected to.

Managing VNet Integration

Connecting and disconnecting with a VNet is at an app level. Operations that can affect the VNet Integration across multiple apps are at the App Service plan level. From the app > Networking > VNet Integration portal, you can get details on your VNet. You can see similar information at the ASP level in the ASP > Networking > VNet Integration portal.

The only operation you can take in the app view of your VNet Integration is to disconnect your app from the VNet it is currently connected to. To disconnect your app from a VNet, select **Disconnect**. Your app will be restarted when you disconnect from a VNet. Disconnecting doesn't change your VNet. The subnet or gateway is not removed. If you then want to delete your VNet, you need to first disconnect your app from the VNet and delete the resources in it such as gateways.

The ASP VNet Integration UI will show you all of the VNet integrations used by the apps in your ASP. To see details on each VNet, click on the VNet you are interested in. There are two actions you can perform here for gateway required VNet Integration.

- **Sync network.** The sync network operation is only for the gateway-dependent VNet Integration feature. Performing a sync network operation ensures that your certificates and network information are in sync. If you add or change the DNS of your VNet, you need to perform a **Sync network** operation. This operation will restart any apps using this VNet.
- **Add routes** Adding routes will drive outbound traffic into your VNet.

Gateway required VNet Integration Routing The routes that are defined in your VNet are used to direct traffic into your VNet from your app. If you need to send additional outbound traffic into the VNet, then you can add those address blocks here. This capability only works with gateway required VNet Integration. Route tables do not affect your app traffic when using gateway required VNet Integration the way that they do with regional VNet Integration.

Gateway required VNet Integration Certificates When the gateway required VNet Integration enabled, there is a required exchange of certificates to ensure the security of the connection. Along with the certificates are the DNS configuration, routes, and other similar things that describe the network. If certificates or network information is changed, you need to click "Sync Network". When you click "Sync Network", you cause a brief outage in connectivity between your app and your VNet. While your app isn't restarted, the loss of connectivity could cause your site to not function properly.

Pricing details

The regional VNet Integration feature has no additional charge for use beyond the ASP pricing tier charges.

There are three related charges to the use of the gateway required VNet Integration feature:

- ASP pricing tier charges - Your apps need to be in a Standard, Premium, or PremiumV2 App Service Plan. You can see more details on those costs here: [App Service Pricing](#).
- Data transfer costs - There is a charge for data egress, even if the VNet is in the same data center. Those charges are described in [Data Transfer Pricing Details](#).
- VPN Gateway costs - There is a cost to the VNet gateway that is required for the point-to-site VPN. The details are on the [VPN Gateway Pricing](#) page.

Troubleshooting

While the feature is easy to set up, that doesn't mean that your experience will be problem free. Should you encounter problems accessing your desired endpoint there are some utilities you can use to test connectivity from the app console. There are two consoles that you can use. One is the Kudu console and the other is the console in the Azure portal. To reach the Kudu console from your app, go to Tools -> Kudu. You can also reach the Kudu console at [sitename].scm.azurewebsites.net. Once the website loads, go to the Debug console tab. To get to the Azure portal hosted console then from your app go to Tools -> Console.

Tools

The tools **ping**, **nslookup**, and **tracert** won't work through the console due to security constraints. To fill the void, two separate tools added. In order to test DNS functionality, we added a tool named nameresolver.exe. The syntax is:

```
nameresolver.exe hostname [optional: DNS Server]
```

You can use **nameresolver** to check the hostnames that your app depends on. This way you can test if you have anything mis-configured with your DNS or perhaps don't have access to your DNS server. You can see the DNS server that your app will use in the console by looking at the environmental variables WEBSITE_DNS_SERVER and WEBSITE_DNS_ALT_SERVER.

The next tool allows you to test for TCP connectivity to a host and port combination. This tool is called **tcpping** and the syntax is:

```
tcpping.exe hostname [optional: port]
```

The **tcpping** utility tells you if you can reach a specific host and port. It only can show success if: there is an application listening at the host and port combination, and there is network access from your app to the specified host and port.

Debugging access to VNet hosted resources

There are a number of things that can prevent your app from reaching a specific host and port. Most of the time it is one of three things:

- **A firewall is in the way.** If you have a firewall in the way, you will hit the TCP timeout. The TCP timeout is 21 seconds in this case. Use the **tcpping** tool to test connectivity. TCP timeouts can be due to many things beyond firewalls but start there.
- **DNS isn't accessible.** The DNS timeout is three seconds per DNS server. If you have two DNS servers, the timeout is 6 seconds. Use nameresolver to see if DNS is working. Remember you can't use nslookup as that doesn't use the DNS your VNet is configured with. If inaccessible, you could have a firewall or NSG blocking access to DNS or it could be down.

If those items don't answer your problems, look first for things like:

regional VNet Integration

- is your destination a non-RFC1918 address and you do not have WEBSITE_VNET_ROUTE_ALL set to 1
- is there an NSG blocking egress from your integration subnet
- if going across ExpressRoute or a VPN, is your on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your VNet but not on-premises, check your routes.
- do you have enough permissions to set delegation on the integration subnet? During regional VNet Integration configuration, your integration subnet will be delegated to Microsoft.Web. The VNet Integration UI will delegate the subnet to Microsoft.Web automatically. If your account does not have sufficient networking permissions to set delegation, you will need someone who can set attributes on your integration subnet to delegate the subnet. To manually delegate the integration subnet, go to the Azure Virtual Network subnet UI and set delegation for Microsoft.Web.

gateway required VNet Integration

- is the point-to-site address range in the RFC 1918 ranges (10.0.0.0-10.255.255.255 / 172.16.0.0-172.31.255.255 / 192.168.0.0-192.168.255.255)?
- Does the Gateway show as being up in the portal? If your gateway is down, then bring it back up.
- Do certificates show as being in sync or do you suspect that the network configuration was changed? If your

certificates are out of sync or you suspect that there has been a change made to your VNet configuration that wasn't synced with your ASPs, then hit "Sync Network".

- if going across a VPN, is the on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your VNet but not on-premises, check your routes.
- are you trying to use a coexistence gateway that supports both point to site and ExpressRoute? Coexistence gateways are not supported with VNet Integration

Debugging networking issues is a challenge because there you cannot see what is blocking access to a specific host:port combination. Some of the causes include:

- you have a firewall up on your host preventing access to the application port from your point to site IP range. Crossing subnets often requires Public access.
- your target host is down
- your application is down
- you had the wrong IP or hostname
- your application is listening on a different port than what you expected. You can match your process ID with the listening port by using "netstat -aon" on the endpoint host.
- your network security groups are configured in such a manner that they prevent access to your application host and port from your point to site IP range

Remember that you don't know what address your app will actually use. It could be any address in the integration subnet or point-to-site address range, so you need to allow access from the entire address range.

Additional debug steps include:

- connect to a VM in your VNet and attempt to reach your resource host:port from there. To test for TCP access, use the PowerShell command **test-netconnection**. The syntax is:

```
test-netconnection hostname [optional: -Port]
```

- bring up an application on a VM and test access to that host and port from the console from your app using **tcpping**

On-premises resources

If your app cannot reach a resource on-premises, then check if you can reach the resource from your VNet. Use the **test-netconnection** PowerShell command to check for TCP access. If your VM can't reach your on-premises resource, your VPN or ExpressRoute connection may not be configured properly.

If your VNet hosted VM can reach your on-premises system but your app can't, then the cause is likely one of the following reasons:

- your routes are not configured with your subnet or point to site address ranges in your on-premises gateway
- your network security groups are blocking access for your Point-to-Site IP range
- your on-premises firewalls are blocking traffic from your Point-to-Site IP range
- you are trying to reach a non-RFC 1918 address using the regional VNet Integration feature

Automation

There is CLI support for regional VNet Integration. To access to the following commands, [Install Azure CLI](#).

```
az webapp vnet-integration --help
```

Group

```
az webapp vnet-integration : Methods that list, add, and remove virtual network integrations from a webapp.
```

This command group is in preview. It may be changed/removed in a future release.

Commands:

```
add      : Add a regional virtual network integration to a webapp.
```

```
list     : List the virtual network integrations on a webapp.
```

```
remove   : Remove a regional virtual network integration from webapp.
```

```
az appservice vnet-integration --help
```

Group

```
az appservice vnet-integration : A method that lists the virtual network integrations used in an appservice plan.
```

This command group is in preview. It may be changed/removed in a future release.

Commands:

```
list    : List the virtual network integrations used in an appservice plan.
```

For gateway required VNet Integration, you can integrate App Service with an Azure Virtual Network using PowerShell. For a ready-to-run script, see [Connect an app in Azure App Service to an Azure Virtual Network](#).

Configure Azure Files in a Windows Container on App Service

2/10/2020 • 2 minutes to read • [Edit Online](#)

NOTE

This article applies to custom Windows containers. To deploy to App Service on *Linux*, see [Serve Content from Azure Storage](#).

This guide shows how to access Azure Storage in Windows Containers. Only [Azure Files Shares](#) and [Premium Files Shares](#) are supported. You use Azure Files Shares in this how-to. Benefits include secured content, content portability, access to multiple apps, and multiple transferring methods.

Prerequisites

- [Azure CLI](#) (2.0.46 or later).
- [An existing Windows Container app in Azure App Service](#)
- [Create Azure file share](#)
- [Upload files to Azure File share](#)

NOTE

Azure Files is non-default storage and billed separately, not included with the web app. It doesn't support using Firewall configuration due to infrastructure limitations.

Limitations

- Azure Storage in Windows containers is **in preview** and **not supported** for **production scenarios**.
- Azure Storage in Windows containers supports mounting **Azure Files containers** (Read / Write) only.
- Azure Storage in Windows containers is currently **not supported** for bringing your own code scenarios on Windows App Service plans.
- Azure Storage in Windows containers **doesn't support** using the **Storage Firewall** configuration because of infrastructure limitations.
- Azure Storage in Windows containers 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 billed independently and **not included** with your web app. Learn more about [Azure Storage pricing](#).

Link storage to your web app (preview)

To mount an Azure Files Share to a directory in your App Service app, you use the `az webapp config storage-account add` command. Storage Type must be AzureFiles.

```
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 of form c:<directory name> >
```

You should do this for any other directories you want to be linked to an Azure Files share.

Verify

Once an Azure Files share 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>
```

Next steps

- [Migrate an ASP.NET app to Azure App Service using a Windows container \(Preview\)](#).

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=bNrVrEFzRHQB17GFJ7boEanetyJ9DGwBSV8OM3Mdh%2FM%3D"
```

If you publish an updated package with the same name to Blob storage, you need to restart your app so that the updated package is loaded into App Service.

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 with a ZIP or WAR file

1/14/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to use a ZIP file or WAR file to deploy your web app to [Azure App Service](#).

This ZIP file deployment uses the same Kudu service that powers continuous integration-based deployments. Kudu supports the following functionality for ZIP file deployment:

- Deletion of files left over from a previous deployment.
- Option to turn on the default build process, which includes package restore.
- Deployment customization, including running deployment scripts.
- Deployment logs.
- A file size limit of 2048 MB.

For more information, see [Kudu documentation](#).

The WAR file deployment deploys your [WAR](#) file to App Service to run your Java web app. See [Deploy WAR file](#).

Prerequisites

To complete the steps in this article, [create an App Service app](#), or use an app that you created for another tutorial.

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

Create a project ZIP 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
```

Deploy ZIP file

In the browser, navigate to https://<app_name>.scm.azurewebsites.net/ZipDeployUI.

Upload the ZIP file you created in [Create a project ZIP file](#) by dragging it to the file explorer area on the web page.

When deployment is in progress, an icon in the top right corner shows you the progress in percentage. The page also shows verbose messages for the operation below the explorer area. When it is finished, the last deployment message should say `Deployment successful`.

The above endpoint does not work for Linux App Services at this time. Consider using FTP or the [ZIP deploy API](#) instead.

Deploy ZIP file with Azure CLI

Deploy the uploaded ZIP file to your web app by using the `az webapp deployment source config-zip` command.

The following example deploys the ZIP file you uploaded. When using a local installation of Azure CLI, specify the path to your local ZIP file for `--src`.

```
az webapp deployment source config-zip --resource-group <group-name> --name <app-name> --src  
clouddrive/<filename>.zip
```

This command deploys the files and directories from the ZIP file to your default App Service application folder (`\home\site\wwwroot`) and restarts the app.

By default, the deployment engine assumes that a ZIP file is ready to run as-is and doesn't run any build automation. To enable the same build automation as in a [Git deployment](#), set the `SCM_DO_BUILD_DURING_DEPLOYMENT` app setting by running the following command in the [Cloud Shell](#):

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings  
SCM_DO_BUILD_DURING_DEPLOYMENT=true
```

For more information, see [Kudu documentation](#).

Deploy ZIP file with REST APIs

You can use the [deployment service REST APIs](#) to deploy the .zip file to your app in Azure. To deploy, send a POST request to `https://<app_name>.scm.azurewebsites.net/api/zipdeploy`. The POST request must contain the .zip file in the message body. The deployment credentials for your app are provided in the request by using HTTP BASIC authentication. For more information, see the [.zip push deployment reference](#).

For the HTTP BASIC authentication, you need your App Service deployment credentials. To see how to set your deployment credentials, see [Set and reset user-level credentials](#).

With cURL

The following example uses the cURL tool to deploy a .zip file. Replace the placeholders `<deployment_user>`, `<zip_file_path>`, and `<app_name>`. When prompted by cURL, type in the password.

```
curl -X POST -u <deployment_user> --data-binary @"<zip_file_path>"  
https://<app_name>.scm.azurewebsites.net/api/zipdeploy
```

This request triggers push deployment from the uploaded .zip file. You can review the current and past deployments by using the `https://<app_name>.scm.azurewebsites.net/api/deployments` endpoint, as shown in the following cURL example. Again, replace `<app_name>` with the name of your app and `<deployment_user>` with the username of your deployment credentials.

```
curl -u <deployment_user> https://<app_name>.scm.azurewebsites.net/api/deployments
```

With PowerShell

The following example uses [Publish-AzWebapp](#) upload the .zip file. Replace the placeholders `<group-name>`, `<app-name>`, and `<zip-file-path>`.

```
Publish-AzWebapp -ResourceGroupName <group-name> -Name <app-name> -ArchivePath <zip-file-path>
```

This request triggers push deployment from the uploaded .zip file.

To review the current and past deployments, run the following commands. Again, replace the `<deployment-user>`, `<deployment-password>`, and `<app-name>` placeholders.

```
$username = "<deployment-user>"  
$password = "<deployment-password>"  
$apiUrl = "https://<app-name>.scm.azurewebsites.net/api/deployments"  
$base64AuthInfo = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes(("{}:{}" -f $username, $password)))  
$userAgent = "powershell/1.0"  
Invoke-RestMethod -Uri $apiUrl -Headers @{Authorization=("Basic {0}" -f $base64AuthInfo)} -UserAgent $userAgent -Method GET
```

Deploy WAR file

To deploy a WAR file to App Service, send a POST request to

`https://<app-name>.scm.azurewebsites.net/api/wardeploy`. The POST request must contain the .war file in the message body. The deployment credentials for your app are provided in the request by using HTTP BASIC authentication.

Always use `/api/wardeploy` when deploying WAR files. This API will expand your WAR file and place it on the shared file drive. using other deployment APIs may result in inconsistent behavior.

For the HTTP BASIC authentication, you need your App Service deployment credentials. To see how to set your deployment credentials, see [Set and reset user-level credentials](#).

With cURL

The following example uses the cURL tool to deploy a .war file. Replace the placeholders `<username>`, `<war-file-path>`, and `<app-name>`. When prompted by cURL, type in the password.

```
curl -X POST -u <username> --data-binary @"<war-file-path>" https://<app-name>.scm.azurewebsites.net/api/wardeploy
```

With PowerShell

The following example uses [Publish-AzWebapp](#) upload the .war file. Replace the placeholders `<group-name>`, `<app-name>`, and `<war-file-path>`.

```
Publish-AzWebapp -ResourceGroupName <group-name> -Name <app-name> -ArchivePath <war-file-path>
```

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

For more advanced deployment scenarios, try [deploying to Azure with Git](#). Git-based deployment to Azure enables version control, package restore, MSBuild, and more.

More resources

- [Kudu: Deploying from a zip file](#)
- [Azure App Service Deployment Credentials](#)

Deploy your app to Azure App Service using FTP/S

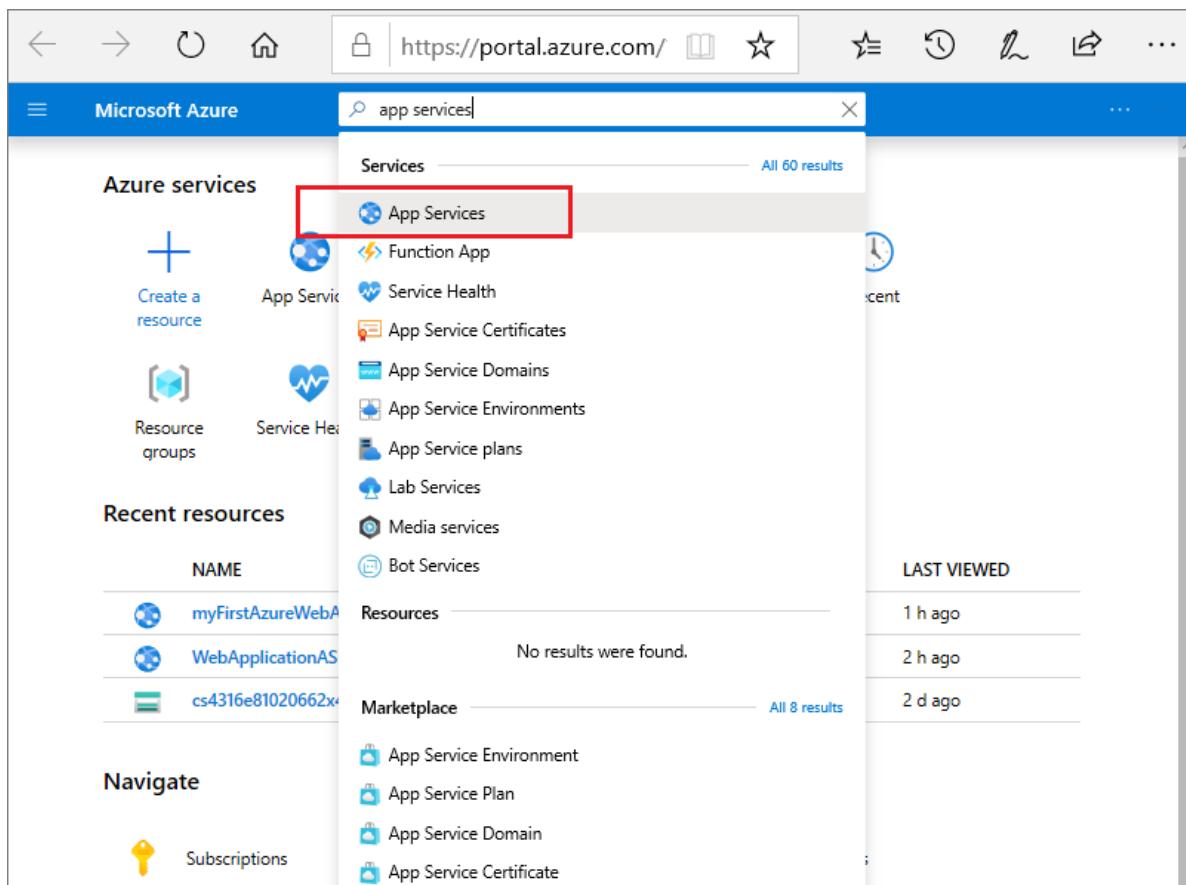
1/6/2020 • 4 minutes to read • [Edit Online](#)

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.

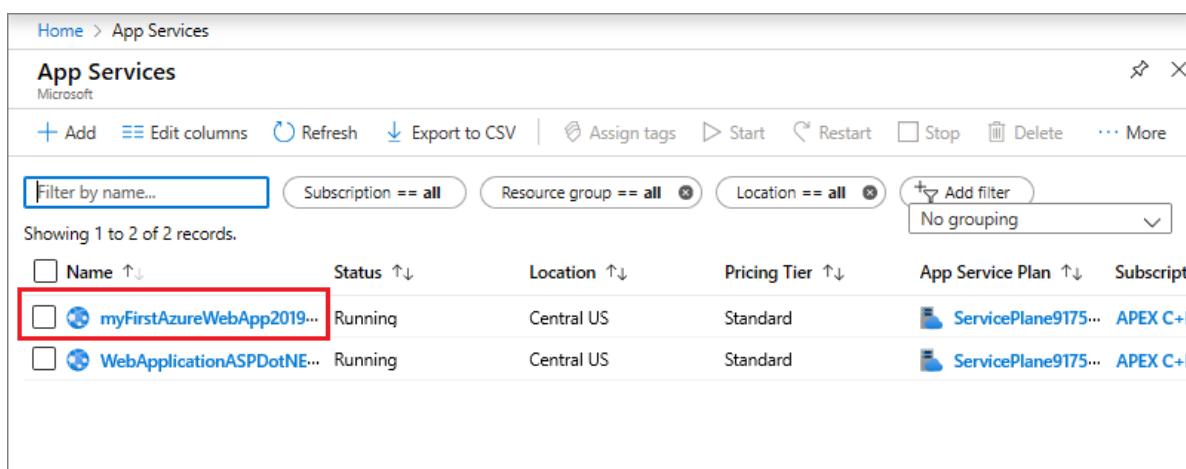
Open FTP dashboard

1. In the [Azure portal](#), search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. The search bar at the top contains the text "app services". Below the search bar, there is a sidebar titled "Azure services" with options like "Create a resource", "Resource groups", and "Recent resources". The main 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", "Bot Services", and "Resources". There is also a section for "Marketplace" with 8 results. 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 in the Azure portal. The top navigation bar includes "Home > App Services". The main area displays a table of deployed apps. The first row, which contains the name "myFirstAzureWebApp2019..." and the status "Running", is highlighted with a red box. The table has columns for Name, Status, Location, Pricing Tier, App Service Plan, and Subscription. There are filter and sorting options at the top of the table.

Name	Status	Location	Pricing Tier	App Service Plan	Subscription
myFirstAzureWebApp2019...	Running	Central US	Standard	ServicePlane9175...	APEX C+L
WebApplicationASPDotNE...	Running	Central US	Standard	ServicePlane9175...	APEX C+L

3. Select **Deployment Center > FTP > Dashboard**.

The screenshot shows the Azure App Service Deployment Center for the 'my-app' app. On the left, a sidebar lists navigation items including Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (with Quickstart and Deployment slots), and Deployment Center (which is selected and highlighted with a red box). Below this is a Settings section with Configuration, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, and TLS/SSL settings. The main area displays several deployment methods: Bitbucket (Configure continuous integration with a Bitbucket repo, Not Authorized), Local Git (Deploy from a local Git repo), OneDrive (Sync content from a OneDrive cloud folder, Not Authorized), Dropbox (Sync content from a Dropbox cloud folder, Not Authorized), External (Deploy from a public Git or Mercurial repo. This option requires manually triggering your deployments), and FTP (Use an FTP connection to access and copy app files, also highlighted with a red box). At the bottom right is a blue 'Dashboard' button.

Get FTP connection information

In the FTP dashboard, select **Copy** to copy the FTPS endpoint and app credentials.

The screenshot shows the 'FTP' dashboard for an app. It features a header with two orange 'FTP' buttons. Below is a message: 'App Service enables you to access your app content through FTP/S. [Learn more](#)'. The 'FTPS Endpoint' field contains the URL 'ftps://waws-prod-am2-263.ftp.azurewebsites.windows.net/' with a red box around the 'Copy' button. Below are tabs for 'App Credentials' (selected) and 'User Credentials'. Under 'App Credentials', there are fields for 'Username' and 'Password', each with its own 'Copy' button highlighted with a red box. A 'Reset Credentials' button is at the bottom. A note above the fields states: 'Application Credentials are auto-generated and provide access only to this specific app or deployment slot. These credentials can be used with FTP, Local Git and WebDeploy. [Learn more](#)'.

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

1. From your FTP client (for example, [Visual Studio](#), [Cyberduck](#), or [WinSCP](#)), use the connection information you gathered to connect to your app.
2. 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).
3. 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](#))

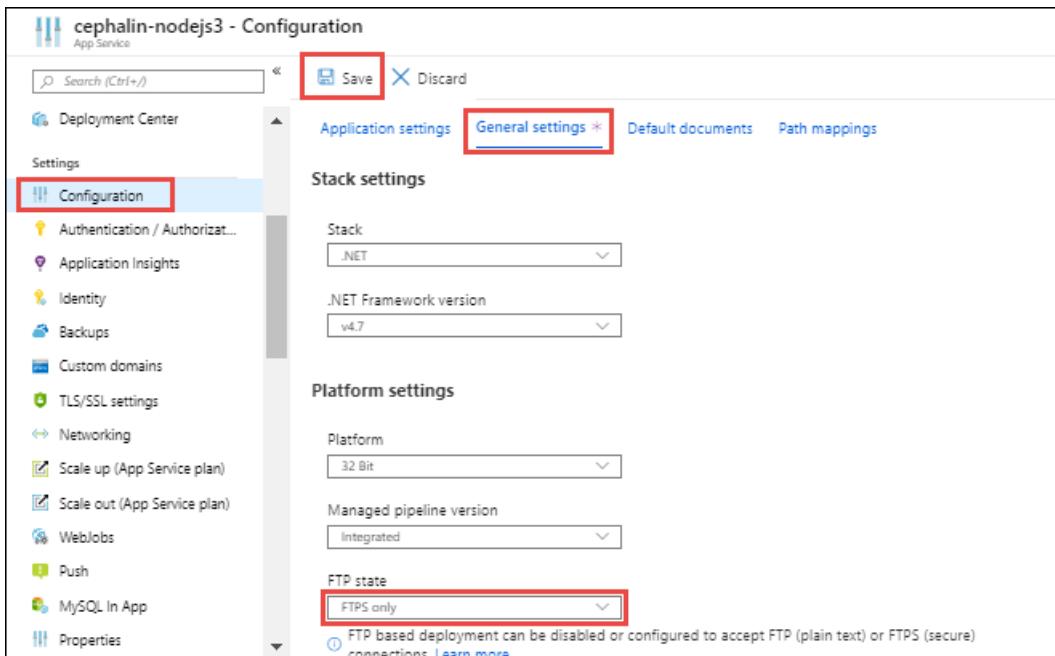
Generate these necessary files manually on your local machine, and then deploy them together with your app.

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.

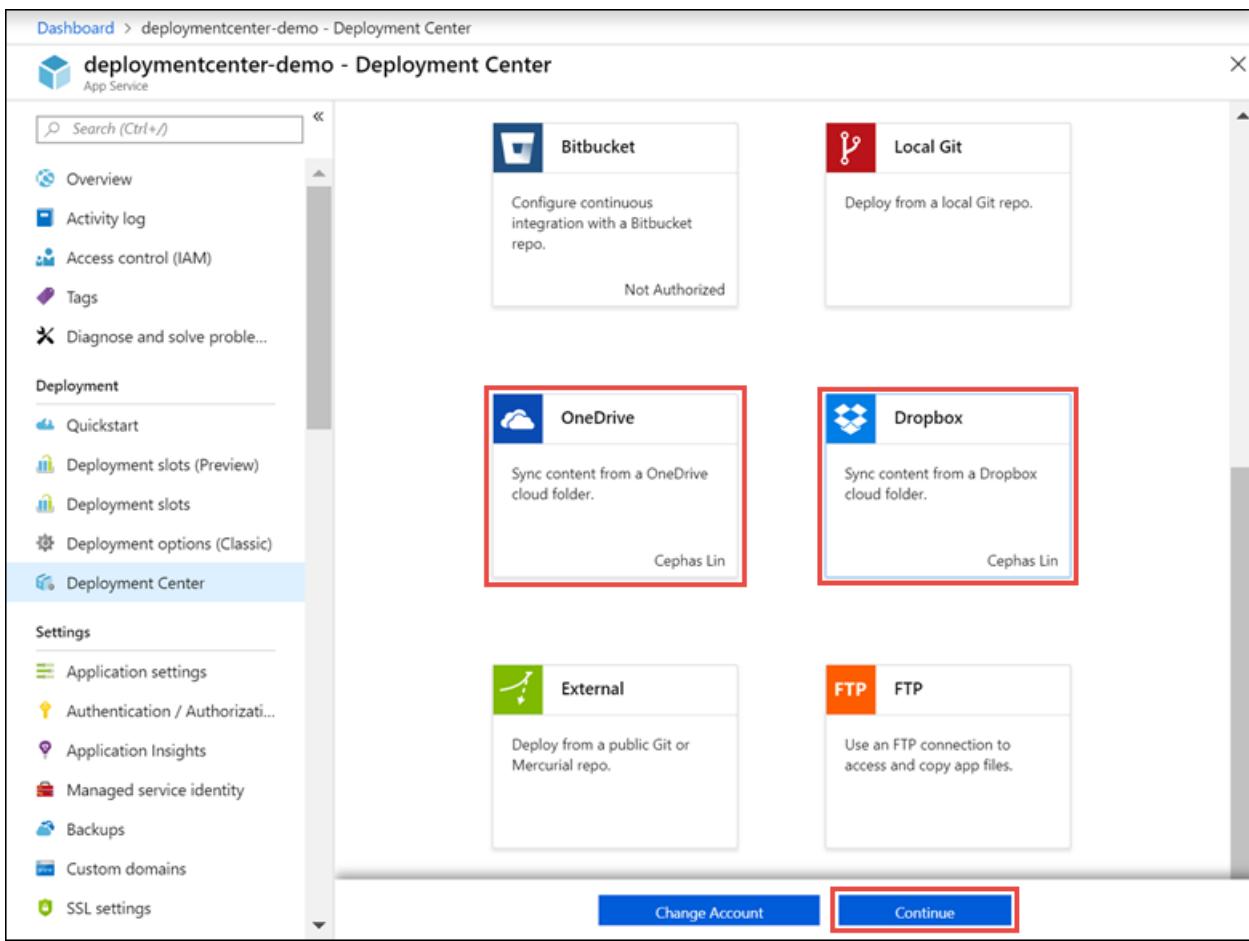
Enable content sync deployment

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. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment slots (Preview), Deployment slots, Deployment options (Classic), and Deployment Center (which is selected and highlighted with a red box). The main area has several cards: Bitbucket (Configure continuous integration with a Bitbucket repo, Not Authorized), Local Git (Deploy from a local Git repo), OneDrive (Sync content from a OneDrive cloud folder, Not Authorized), Dropbox (Sync content from a Dropbox cloud folder, Not Authorized), External (Deploy from a public Git or Mercurial repo), and FTP (Use an FTP connection to access and copy app files). At the bottom right, there's a large blue 'Authorize' button.

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**.



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

Source: Dropbox Folder: /Apps/Azure/deploymentcenter-demo

Build: Kudu Rollback Enabled: No

Deployment

- Quickstart
- Deployment slots (Preview)
- Deployment slots
- Deployment options (Classic)
- Deployment Center**

Settings

- Application settings
- Authentication / Authorization
- Application Insights
- Managed service identity
- Backups
- Custom domains
- SSL settings

TIME	STATUS	COMMIT ID (AUTI)	CHECKIN MESSAGE	LOGS
Monday, December 3, 2018	Success (Active)	8d9ee47 (Cephas L)	Synchronized 0 change(s) from Dropbox.	

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 includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart, Deployment slots (Preview), Deployment slots, Deployment options (Classic)), Deployment Center (selected and highlighted with a red box), Settings (Application settings, Authentication / Authorization, Application Insights, Managed service identity, Backups, Custom domains, SSL settings). The main content area displays deployment details: Source is Dropbox, Build is Kudu, Folder is /Apps/Azure/deploymentcenter-demo, Rollback Enabled is No. It also shows a deployment log entry from Monday, December 3, 2018, at 10:51:46 AM GMT+, status Success (Active), commit ID 8d9ee47, and a message indicating synchronization from Dropbox.

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 to Azure App Service

1/6/2020 • 6 minutes to read • [Edit Online](#)

Azure App Service enables continuous deployment from GitHub, BitBucket, and [Azure Repos](#) repositories by pulling in the latest updates. This article shows you how to use the Azure portal to continuously deploy your app through the Kudu build service or [Azure Pipelines](#).

For more information on the source control services, see [Create a repo \(GitHub\)](#), [Create a repo \(BitBucket\)](#), or [Create a new Git repo \(Azure Repos\)](#).

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><job_name>/run.<extension></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 Kudu WebJobs documentation .
Functions	See Continuous deployment for Azure Functions .

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.

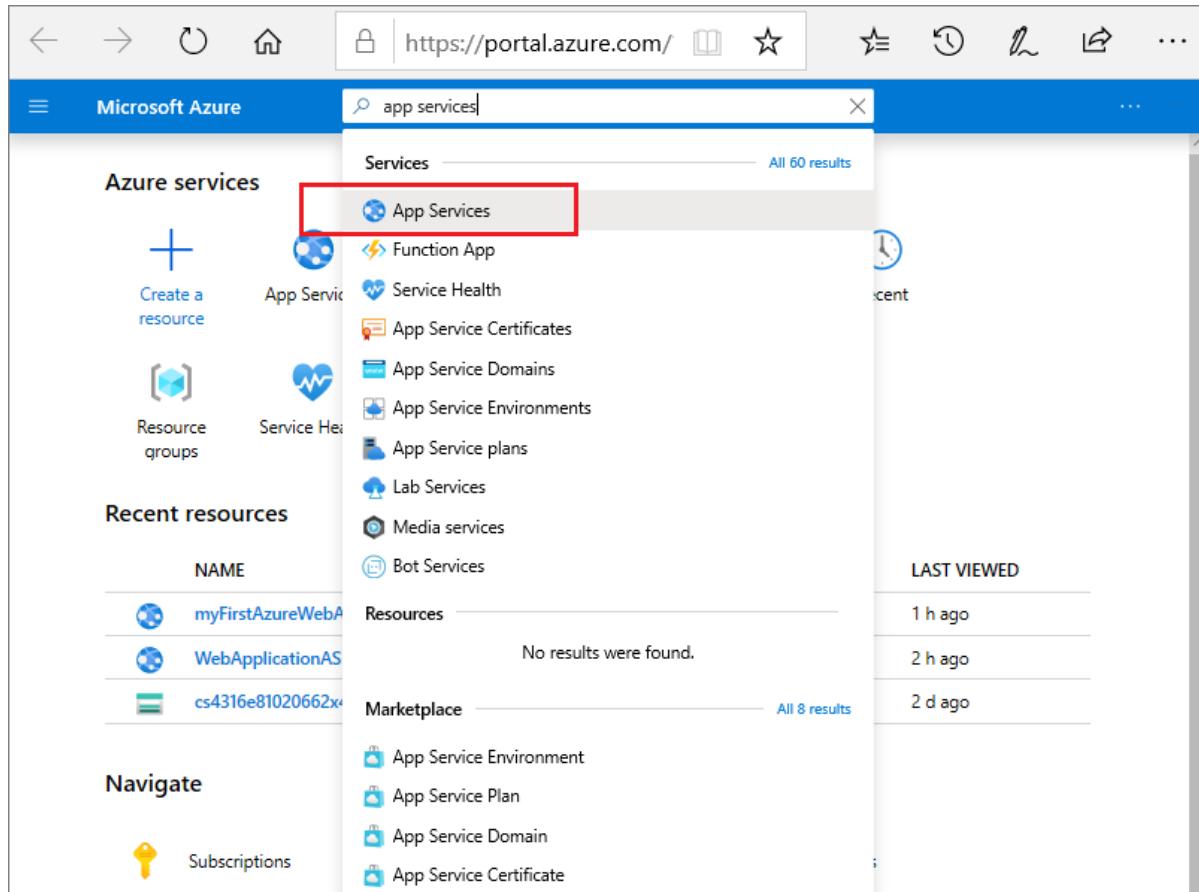
Authorize Azure App Service

To use Azure Repos, make sure your Azure DevOps organization is linked to your Azure subscription. For more

information, see [Set up an Azure DevOps Services account so it can deploy to a web app](#).

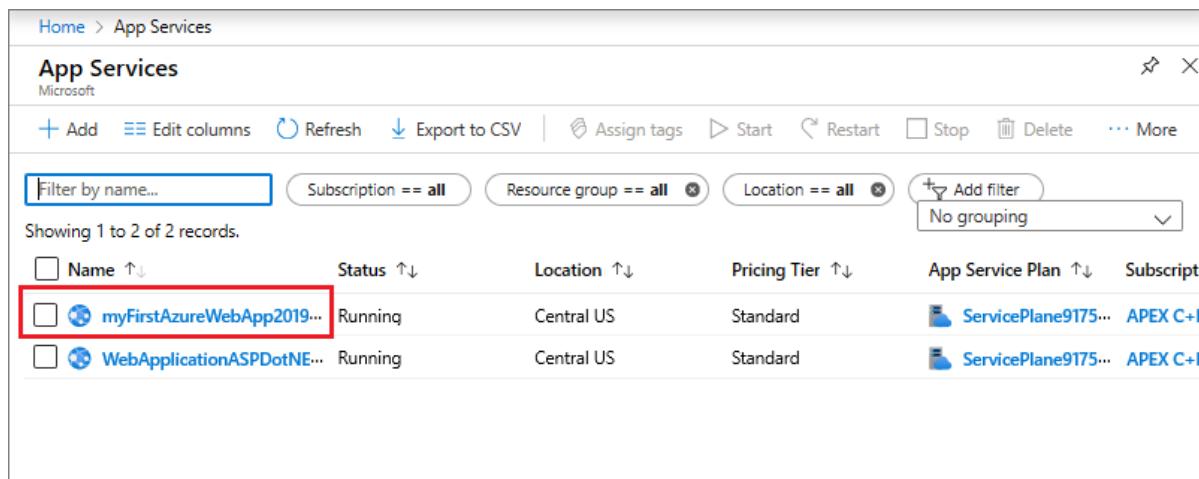
For Bitbucket or GitHub, authorize Azure App Service to connect to your repository. You only need to authorize with a source control service once.

1. In the [Azure portal](#), search for and select **App Services**.



The screenshot shows the Microsoft Azure portal homepage. The search bar at the top contains the text "app services". On the left sidebar, under "Azure services", there is a section for "App Services" which is highlighted with a red box. Other items in the sidebar include "Create a resource", "Resource groups", "Recent resources" (listing "myFirstAzureWebA", "WebApplicationAS", and "cs4316e81020662x"), and "Navigate" (listing "Subscriptions"). The main content area shows a search results page for "Services" with "All 60 results". The first result, "App Services", is also 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 sections for "Resources" and "Marketplace". A "LAST VIEWED" column on the right lists "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 in the Azure portal. The top navigation bar includes "Home > App Services" and "Microsoft". Below the navigation are buttons for "+ Add", "Edit columns", "Refresh", "Export to CSV", "Assign tags", "Start", "Restart", "Stop", "Delete", and "More". There are filters for "Filter by name...", "Subscription == all", "Resource group == all", "Location == all", and "Add filter" (with a dropdown set to "No grouping"). The main table displays two records:

Name	Status	Location	Pricing Tier	App Service Plan	Subscription
myFirstAzureWebApp2019...	Running	Central US	Standard	ServicePlane9175...	APEX C+L
WebApplicationASPDotNE...	Running	Central US	Standard	ServicePlane9175...	APEX C+L

3. On the app page, select **Deployment Center** in the left menu.

4. On the **Deployment Center** page, select **GitHub** or **Bitbucket**, and then select **Authorize**.

The screenshot shows the 'Deployment Center' page for an 'App Service' named 'myFirstAzureWebApp20190516094611'. The left sidebar has a red box around the 'Deployment Center' link under 'Deployment slots'. The main area shows four steps: SOURCE CONTROL (1), BUILD PROVIDER (2), CONFIGURE (3), and SUMMARY (4). Step 1 is highlighted with a blue circle. Under 'Continuous Deployment (CI / CD)', there are four options: 'Azure Repos', 'GitHub', 'Bitbucket', and 'Local Git'. The 'GitHub' option is highlighted with a red box and has a 'Not Authorized' message. A large red box surrounds the 'Authorize' button at the bottom.

5. Sign in to the service if necessary, and follow the authorization prompts.

Enable continuous deployment

After you authorize a source control service, configure your app for continuous deployment through the built-in [Kudu App Service build server](#), or through [Azure Pipelines](#).

Option 1: Use the App Service build service

You can use the built-in Kudu App Service build server to continuously deploy from GitHub, Bitbucket, or Azure Repos.

1. In the [Azure portal](#), search for and select **App Services**, and then select the web app you want to deploy.
2. On the app page, select **Deployment Center** in the left menu.
3. Select your authorized source control provider on the **Deployment Center** page, and select **Continue**. For GitHub or Bitbucket, you can also select **Change account** to change the authorized account.

NOTE

To use Azure Repos, make sure your Azure DevOps Services organization is linked to your Azure subscription. For more information, see [Set up an Azure DevOps Services account so it can deploy to a web app](#).

4. For GitHub or Azure Repos, on the **Build provider** page, select **App Service build service**, and then select **Continue**. Bitbucket always uses the App Service build service.

Home > App Services > myFirstAzureWebApp - Deployment Center

myFirstAzureWebApp - Deployment Center

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Deployment

Quickstart

Deployment slots

Deployment Center

Settings

Configuration

Application settings (Classic)

Authentication / Authorization

Application Insights

Identity

Backups

Custom domains

TLS/SSL settings

Networking

Scale up (App Service plan)

Deployment Center

App Service Deployment Center enables you to choose the location of your code as well as options for build and deployment to the cloud. [Learn more](#)

SOURCE CONTROL 2 BUILD PROVIDER 3 CONFIGURE 4 SUMMARY

App Service build service

Use App Service as the build server. The App Service Kudu engine will automatically build your code during deployment when applicable with no additional configuration required.

Azure Pipelines (Preview)

Configure a robust deployment pipeline for your application using Azure Pipelines, part of Azure DevOps Services (formerly known as VSTS). The

Back Continue

5. On the **Configure** page:

- For GitHub, drop down and select the **Organization**, **Repository**, and **Branch** you want to deploy continuously.

NOTE

If you don't see any repositories, you may need to authorize Azure App Service in GitHub. Browse to your GitHub repository and go to **Settings > Applications > Authorized OAuth Apps**. Select **Azure App Service**, and then select **Grant**. For organization repositories, you must be an owner of the organization to grant the permissions.

- For Bitbucket, select the Bitbucket **Team**, **Repository**, and **Branch** you want to deploy continuously.
- For Azure Repos, select the **Azure DevOps Organization**, **Project**, **Repository**, and **Branch** you want to deploy continuously.

NOTE

If your Azure DevOps organization isn't listed, make sure it's linked to your Azure subscription. For more information, see [Set up an Azure DevOps Services account so it can deploy to a web app..](#)

6. Select **Continue**.

The screenshot shows the Azure Deployment Center interface. On the left, there's a navigation sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings (Configuration, Application settings (Classic), Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, Networking, Scale up (App Service plan)), and Home > App Services > myFirstAzureWebApp - Deployment Center.

The main area is titled "Deployment Center" with a sub-header: "App Service Deployment Center enables you to choose the location of your code as well as options for build and deployment to the cloud. Learn more".

A progress bar at the top indicates the deployment process: SOURCE CONTROL (green checkmark), BUILD PROVIDER (green checkmark), CONFIGURE (blue circle with '3'), and SUMMARY (grey circle with '4').

Code section:

- Organization: test
- Repository: docs
- Branch: live

A note says: "If you can't find an organization or repository, you might need to enable additional permissions on GitHub." A "Back" button and a "Continue" button are at the bottom, with the "Continue" button highlighted by a red box.

7. After you configure the build provider, review the settings on the **Summary** page, and then select **Finish**.

New commits in the selected repository and branch now deploy continuously into your App Service app. You can track the commits and deployments on the **Deployment Center** page.

The screenshot shows the Azure Deployment Center interface. The left sidebar is identical to the previous screenshot. The main area shows deployment details:

- Source: GitHub (<https://github.com/test/docs>)
- Build: Kudu
- Repository: https://github.com/test/docs
- Branch: live
- Source Control Type: Git

A table shows deployment logs:

TIME	STATUS	COMMIT ID (AUTH)	CHECKIN MESSAGE	LOGS
Monday, June 10, 2019	Pending	temp-99 (N/A)	Fetch from https://github.com/test/docs.git	

Option 2: Use Azure Pipelines

If your account has the necessary permissions, you can set up Azure Pipelines to continuously deploy from GitHub or Azure Repos repositories. For more information about deploying through Azure Pipelines, see [Deploy a web app to Azure App Services](#).

For Azure App Service to create continuous delivery Azure Pipelines in your Azure DevOps organization:

- 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 configure Azure Pipelines (Preview):

1. In the [Azure portal](#), search for and select **App Services**, and then select the web app you want to deploy.
2. On the app page, select **Deployment Center** in the left menu.
3. On the **Build provider** page, select **Azure Pipelines (Preview)**, and then select **Continue**.
4. On the **Configure** page, in the **Code** section:
 - For GitHub, drop down and select the **Organization**, **Repository**, and **Branch** you want to deploy continuously.

NOTE

If you don't see any repositories, you may need to authorize Azure App Service in GitHub. Browse to your GitHub repository and go to **Settings > Applications > Authorized OAuth Apps**. Select **Azure App Service**, and then select **Grant**. For organization repositories, you must be an owner of the organization to grant the permissions.

- For Azure Repos, select the **Azure DevOps Organization**, **Project**, **Repository**, and **Branch** you want to deploy continuously, or configure a new Azure DevOps organization.

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](#).

5. Select **Continue**.
6. For Azure Repos, in the **Build** section, specify the language framework that Azure Pipelines should use to run build tasks, and then select **Continue**.
7. On the **Test** page, choose whether to enable load tests, and then select **Continue**.
8. 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**.

NOTE

Azure Pipelines doesn't allow continuous delivery to the production slot. This restriction prevents accidental deployments to production. Set up continuous delivery to a staging slot, verify the changes there, and then swap the slots when you are ready.

9. After you configure the build provider, review the settings on the **Summary** page, and then select **Finish**.

New commits in the selected repository and branch now deploy continuously into your App Service app. You can track the commits and deployments on the **Deployment Center** page.

Home > App Services > myFirstAzureWebApp - Deployment Center

myFirstAzureWebApp - Deployment Center

Search (Ctrl+ /) Refresh Disconnect Sync FTP/Credentials

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Security Deployment Quickstart Deployment slots Deployment Center

Source GitHub Repository https://github.com/test/docs

Build Kudu Branch live Source Control Type Git

TIME	STATUS	COMMIT ID (AUTH)	CHECKIN MESSAGE	LOGS
Monday, June 10, 2019	Pending	temp-99 (N/A)	Fetch from https://github.com/test/docs.git	
11:16:27 AM GMT-7	Pending	temp-99 (N/A)	Fetch from https://github.com/test/docs.git	

Disable continuous deployment

To disable continuous deployment, select **Disconnect** at the top of your app's **Deployment Center** page.

Home > App Services > myFirstAzureWebApp - Deployment Center

myFirstAzureWebApp - Deployment Center

Search (Ctrl+ /) Refresh Disconnect Sync FTP/Credentials

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Security Deployment Quickstart Deployment slots Deployment Center

Source GitHub Repository https://github.com/test/docs

Build Kudu Branch live Source Control Type Git

TIME	STATUS	COMMIT ID (AUTH)	CHECKIN MESSAGE	LOGS
Monday, June 10, 2019	Pending	temp-99 (N/A)	Fetch from https://github.com/test/docs.git	
11:16:27 AM GMT-7	Pending	temp-99 (N/A)	Fetch from https://github.com/test/docs.git	

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.

Use unsupported repos

For Windows apps, you can manually configure continuous deployment from a cloud Git or Mercurial repository that the portal doesn't directly support, such as [GitLab](#). You do it by choosing the External box in the **Deployment Center** page. For more information, see [Set up continuous deployment using manual steps](#).

Additional resources

- Investigate common issues with continuous deployment
- Use Azure PowerShell
- Git documentation
- Project Kudu

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><job_name>/run.<extension></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 Kudu WebJobs documentation .
Functions	See Continuous deployment for Azure Functions .

To customize your deployment, include a `.deployment` file in the repository root. For more information, see [Customize deployments](#) and [Custom deployment script](#).

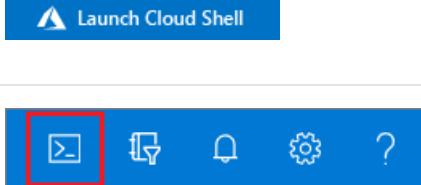
NOTE

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

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

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

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.

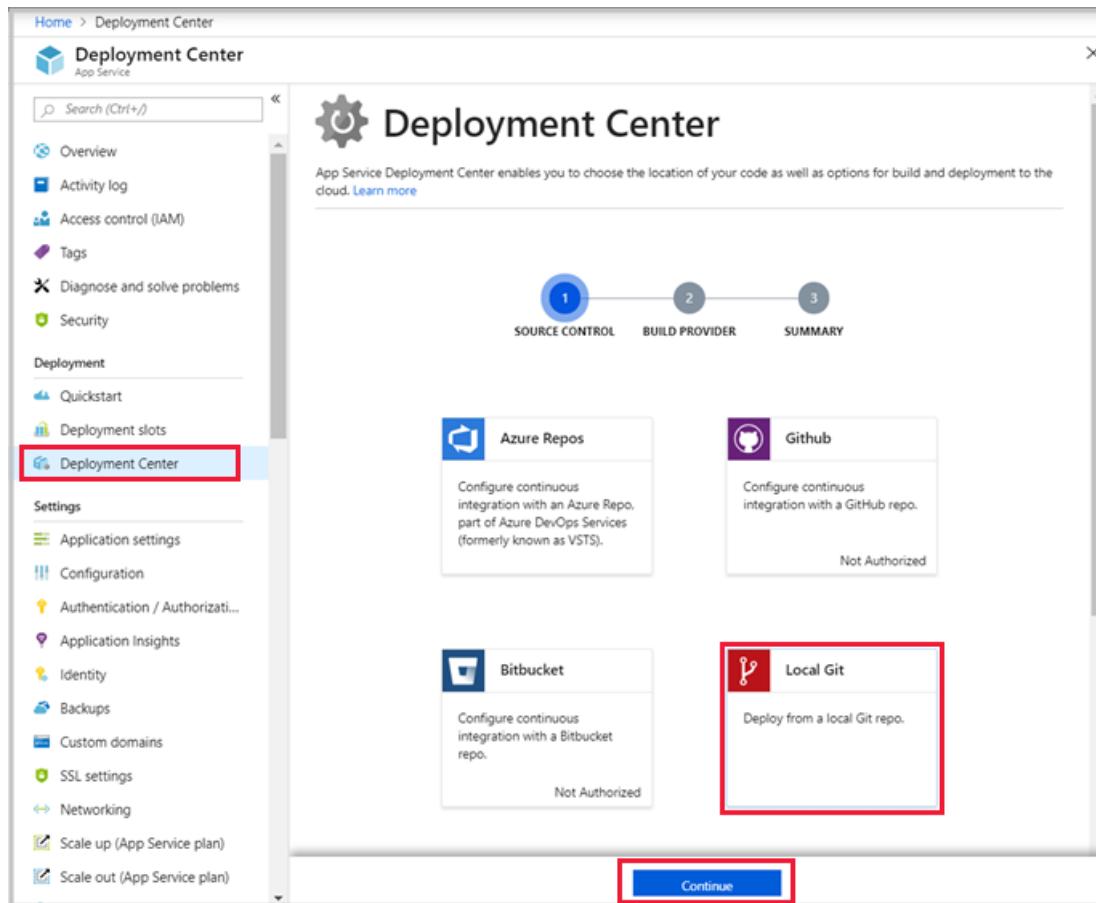
Deploy with Azure Pipelines builds

If your account has the necessary permissions, you can set up Azure Pipelines (Preview) to enable local Git deployment for your app.

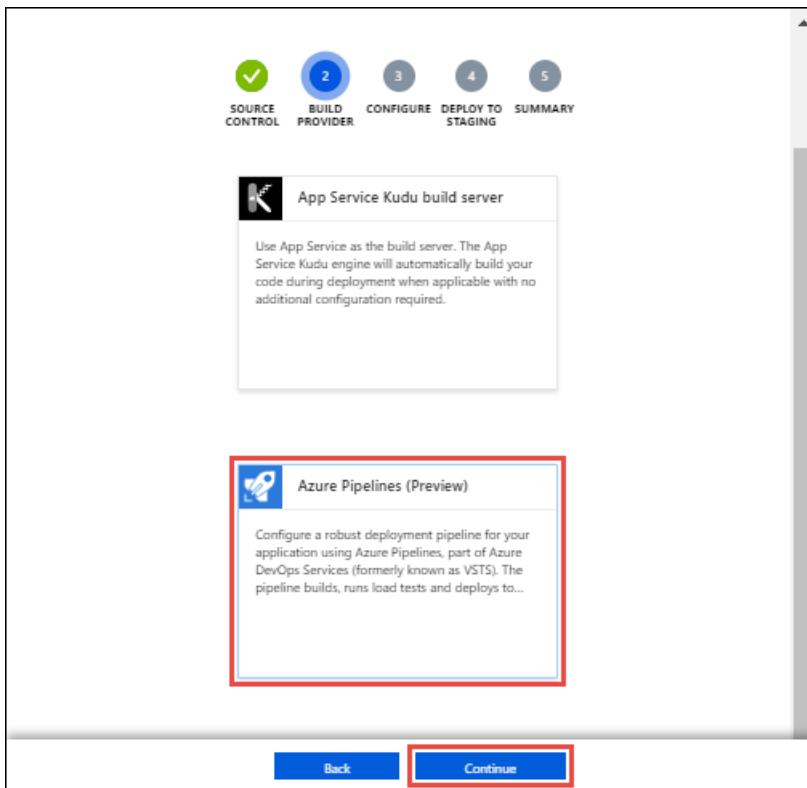
- Your Azure account must have permissions to write to Azure Active Directory and create a service.
- Your Azure account must have the **Owner** role in your Azure subscription.
- You must be an administrator in the Azure DevOps project you want to use.

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), Branch (master), Slot (Production). Below this, a deployment history entry for Monday, March 4, 2019, shows a green checkmark indicating successful setup of Continuous Delivery for the repository, with links to Build Pipeline, Release Pipeline, and View Instructions. The timestamp is 4:29:53 PM GMT-6.

- 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>
```

- Push to the Azure remote with `git push azure master`.
- On the **Git Credential Manager** page, sign in with your visualstudio.com username. For other authentication methods, see [Azure DevOps Services authentication overview](#).
- Once deployment is finished, view the build progress at https://<azure_devops_account>.visualstudio.com/<project_name>/_build, and the deployment progress at https://<azure_devops_account>.visualstudio.com/<project_name>/_release.
- 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>Malformed package.json file: <code>npm ERR! Couldn't read dependencies.</code></p> <p>Native module doesn't have a binary distribution for Windows: <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 to App Service using GitHub Actions

2/21/2020 • 5 minutes to read • [Edit Online](#)

[GitHub Actions](#) gives you the flexibility to build an automated software development lifecycle workflow. With the Azure App Service Actions for GitHub, you can automate your workflow to deploy to [Azure 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 workflow, the file has three sections:

SECTION	TASKS
Authentication	1. Define a service principal 2. Create a GitHub secret
Build	1. Set up the environment 2. Build the web app
Deploy	1. Deploy the web app

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/<group-name>/providers/Microsoft.Web/sites/<app-name> --sdk-auth
```

In this example, replace the placeholders in the resource with your subscription ID, resource group name, and app name. The output is the role assignment credentials that provide access to your App Service app. Copy this JSON object, which you can use to authenticate from GitHub.

NOTE

You do not need to create a service principal if you decide to use publish profile for authentication.

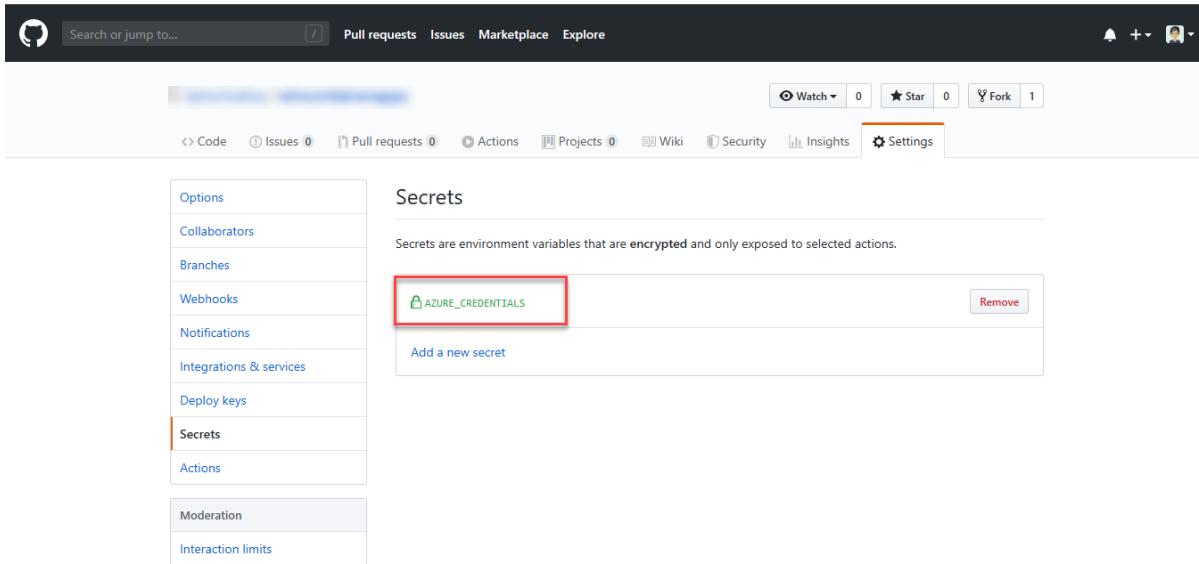
IMPORTANT

It is always a good practice to grant minimum access. This is why the scope in the previous example is limited to the specific App Service app and not the entire resource group.

Configure the GitHub secret

You could also use app-level credentials i.e. publish profile for deployment. Follow the steps to configure the secret:

1. Download the publish profile for the App Service app from the portal using **Get Publish profile** option.
2. In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**

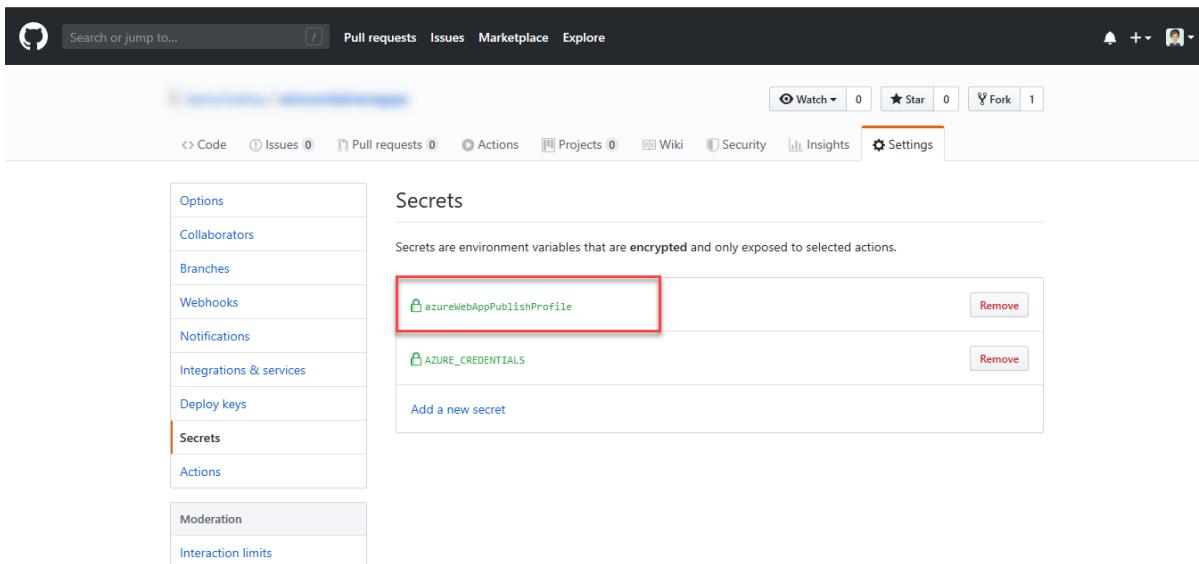


The screenshot shows the GitHub repository settings page with the 'Secrets' tab selected. On the left, a sidebar lists options like Collaborators, Branches, Webhooks, Notifications, Integrations & services, Deploy keys, **Secrets** (which is selected and highlighted in orange), and Actions. The main area is titled 'Secrets' and contains a note: 'Secrets are environment variables that are encrypted and only exposed to selected actions.' Below this is a table with one row. The first column shows a lock icon and the value 'AZURE_CREDENTIALS'. The second column has a 'Remove' button. At the bottom of the table is a link 'Add a new secret'.

3. Paste the contents for the downloaded publish profile file into the secret's value field.
4. Now in the workflow file in your branch: `.github/workflows/workflow.yml` replace the secret for the input `publish-profile` of the deploy Azure Web App action.

```
- uses: azure/webapps-deploy@v1
  with:
    creds: ${{ secrets.azureWebAppPublishProfile }}
```

5. You see the secret as shown below once defined.



The screenshot shows the GitHub repository settings page with the 'Secrets' tab selected. The sidebar and table structure are identical to the previous screenshot, but the secret value has changed. The first column now shows a lock icon and the value 'azureWebAppPublishProfile'. The second column still has a 'Remove' button. The 'Add a new secret' link remains at the bottom.

Set up the environment

Setting up the environment can be done using one of the setup actions.

LANGUAGE	SETUP ACTION
.NET	<code>actions/setup-dotnet</code>
Java	<code>actions/setup-java</code>
JavaScript	<code>actions/setup-node</code>
Python	<code>actions/setup-python</code>

The following examples show the part of the workflow that sets up the environment for the various supported languages:

JavaScript

```
- name: Setup Node 10.x
uses: actions/setup-node@v1
with:
  node-version: '10.x'
```

Python

```
- name: Setup Python 3.6
uses: actions/setup-python@v1
with:
  python-version: 3.6
```

.NET

```
- name: Setup Dotnet 2.2.300
uses: actions/setup-dotnet@v1
with:
  dotnet-version: '2.2.300'
```

Java

```
- name: Setup Java 1.8.x
uses: actions/setup-java@v1
with:
  # If your pom.xml <maven.compiler.source> version is not in 1.8.x
  # Please change the Java version to match the version in pom.xml <maven.compiler.source>
  java-version: '1.8.x'
```

Build the web app

This depends on the language and for languages supported by Azure App Service, this section should be the standard build steps of each language.

The following examples show the part of the workflow that builds the web app, in the various supported languages.

JavaScript

```

- name: 'Run npm'
  shell: bash
  run: |
    # If your web app project is not located in your repository's root
    # Please change your directory for npm in pushd
    pushd .
    npm install
    npm run build --if-present
    npm run test --if-present
  popd

```

Python

```

- name: 'Run pip'
  shell: bash
  run: |
    # If your web app project is not located in your repository's root
    # Please change your directory for pip in pushd
    pushd .
    python -m pip install --upgrade pip
    pip install -r requirements.txt --target=".python_packages/lib/python3.6/site-packages"
  popd

```

.NET

```

- name: 'Run dotnet build'
  shell: bash
  run: |
    # If your web app project is not located in your repository's root
    # Please consider using pushd to change your path
    pushd .
    dotnet build --configuration Release --output ./output
  popd

```

Java

```

- uses: actions/checkout@v1
- name: Set up JDK 1.8
  uses: actions/setup-java@v1
  with:
    java-version: 1.8
- name: Build with Maven
  run: mvn -B package --file pom.xml

```

Deploy to App Service

To deploy your code to an App Service app, use the `azure/webapps-deploy@v1` action. This action has four parameters:

PARAMETER	EXPLANATION
app-name	(Required) Name of the App Service app
publish-profile	(Optional) Publish profile file contents with Web Deploy secrets

PARAMETER	EXPLANATION
package	(Optional) Path to package or folder. *.zip, *.war, *.jar or a folder to deploy
slot-name	(Optional) Enter an existing Slot other than the Production slot

Deploy using Publish Profile

Below is the sample workflow to build and deploy a Node.js app to Azure using publish profile.

```
# File: .github/workflows/workflow.yml

on: push

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      # checkout the repo
      - name: 'Checkout GitHub Action'
        uses: actions/checkout@master

      - name: Setup Node 10.x
        uses: actions/setup-node@v1
        with:
          node-version: '10.x'
      - name: 'npm install, build, and test'
        run: |
          npm install
          npm run build --if-present
          npm run test --if-present

      - name: 'Run Azure webapp deploy action using publish profile credentials'
        uses: azure/webapps-deploy@v1
        with:
          app-name: node-rn
          publish-profile: ${{ secrets.azureWebAppPublishProfile }}
```

Deploy using Azure service principal

Below is the sample workflow to build and deploy a Node.js app to Azure using an Azure service principal.

```
on: [push]

name: Node.js

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      # checkout the repo
      - name: 'Checkout GitHub Action'
        uses: actions/checkout@master

      - uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - name: Setup Node 10.x
        uses: actions/setup-node@v1
        with:
          node-version: '10.x'

      - name: 'npm install, build, and test'
        run: |
          npm install
          npm run build --if-present
          npm run test --if-present

      # deploy web app using Azure credentials
      - uses: azure/webapps-deploy@v1
        with:
          app-name: 'node-rn'

      # Azure logout
      - name: 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.

- [Actions workflow to deploy to Azure](#)
- [Azure login](#)
- [Azure WebApp](#)
- [Azure WebApp for containers](#)
- [Docker login/logout](#)
- [Events that trigger workflows](#)
- [K8s deploy](#)
- [Starter Workflows](#)

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

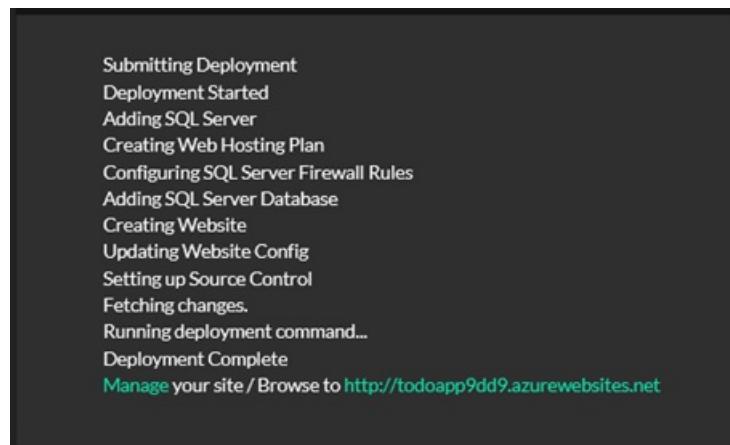
So now let's get right to it.

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

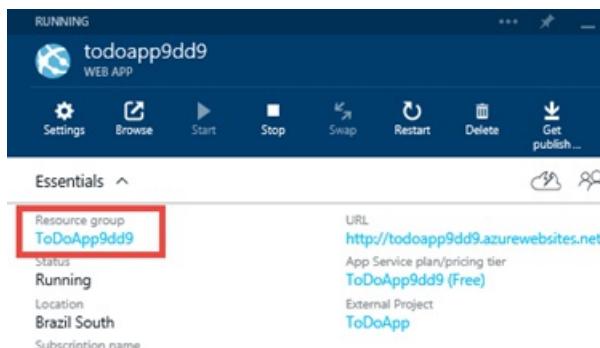
Directory	Subscription
Default Directory	Visual Studio Ultimate with MSDN
Resource Group	Resource Group Name
Create New	ToDoApp9dd9
Site Name <small>Name is available</small>	Site Location
ToDoApp9dd9	Brazil South
Sku	Sql Server Name
Free	todoapp9dd9-server
Sql Server Location	Sql Server Admin Login
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	910b4fc8-6a29-4c3e-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.

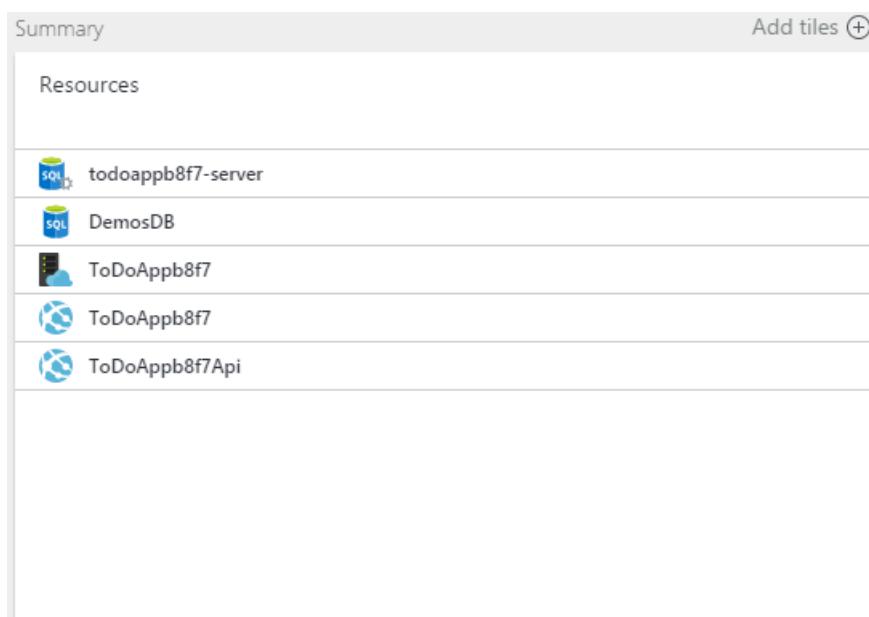


The UI would be a little slow when you first browse to it because the apps are just starting up, but convince yourself that it's a fully-functional application.

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:

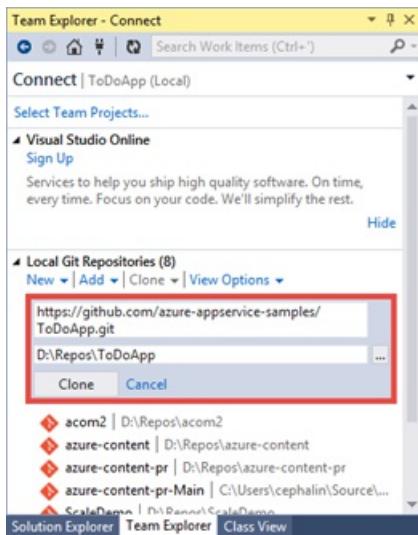
- The Deploy to Azure button
- azuredeploy.json in the repo root

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.

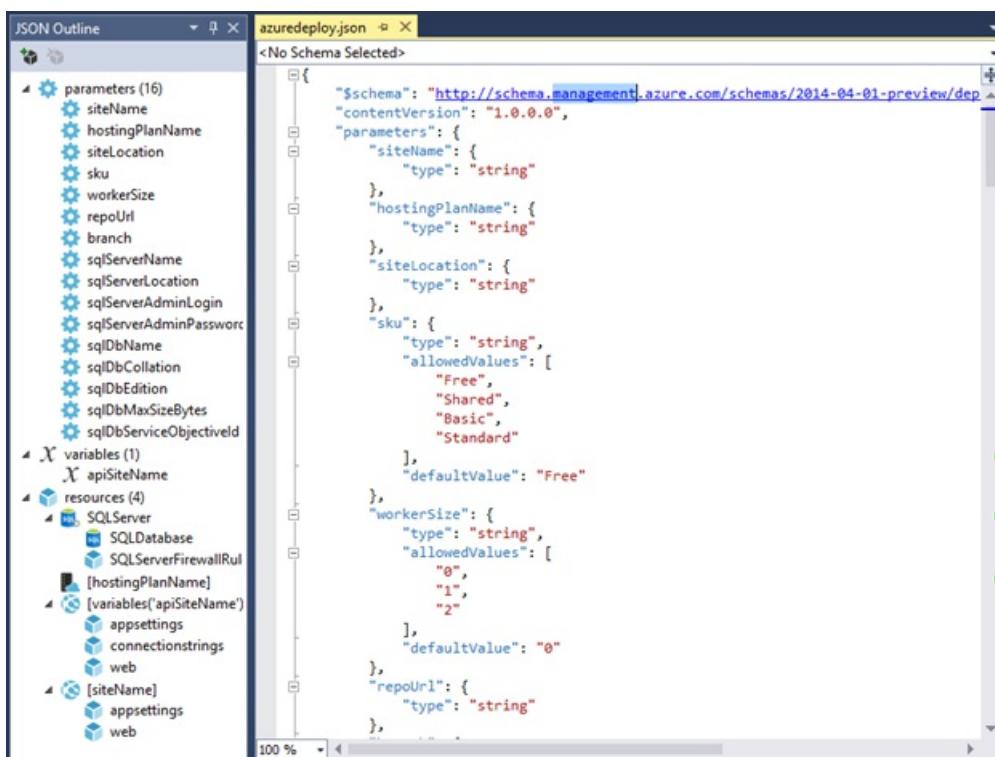
Examine (or edit) AZUREDEPLOY.JSON

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), SQLDatabase, SQLServerFirewallRule, [hostingPlanName] (highlighted in red), [variables('apiSiteName')], and [siteName]. The code editor on the right shows the corresponding JSON code:

```

    "hostingPlanName": {
      "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 (highlighted in red), SQLDatabase, SQLServerFirewallRule, [hostingPlanName], [variables('apiSiteName')], and [siteName]. The code editor on the right shows the corresponding JSON code:

```

  "SQLServer": {
    "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:

```

1 {
2   "id": "/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/sites/ToDoApp9dd9/config/appsettings",
3   "name": "appsettings",
4   "type": "Microsoft.Web/sites/config",
5   "kind": null,
6   "location": "Brazil South",
7   "tags": {
8     "hidden-related:/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/serverfarms/ToDoApp9dd9": "Resource"
9   },
10  "plan": null,
11  "properties": [
12    {
13      "name": "PROJECT",
14      "value": "src\\MultiChannelToDo\\MultiChannelToDo.csproj"
15    },
16    {
17      "name": "clientUrl",
18      "value": "http://ToDoApp9dd9.azurewebsites.net"
19    }
20  ]

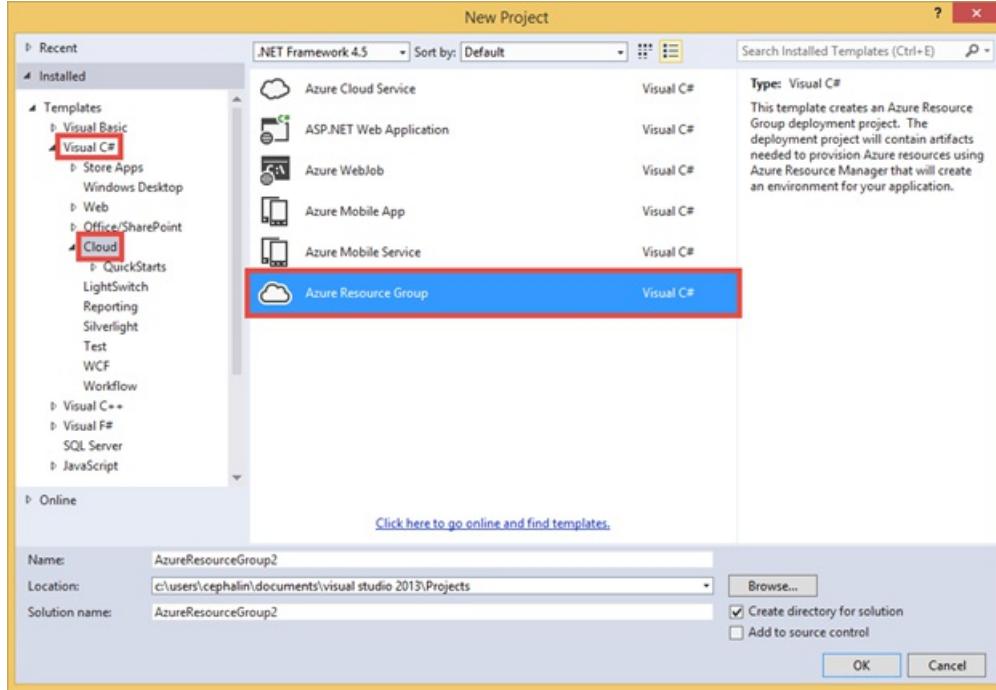
```

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.

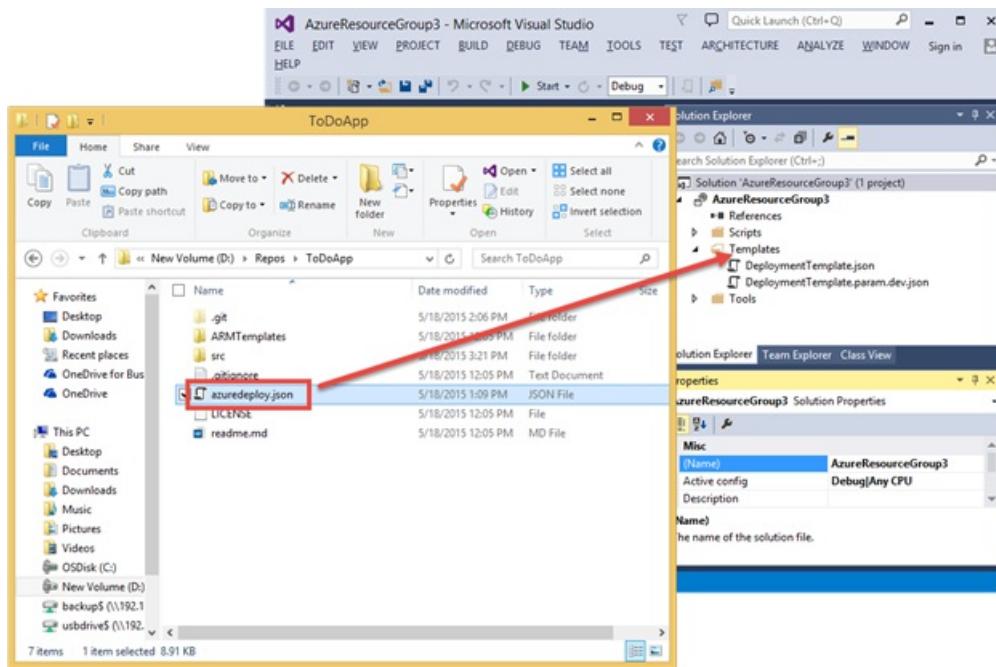
Deploy the resource group template yourself

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:

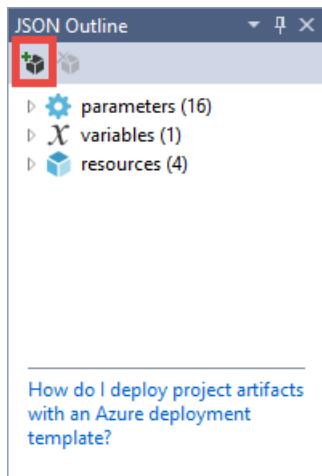
1. In Visual Studio, click **File > New > Project**.
2. Click **Visual C# > Cloud > Azure Resource Group**, then click **OK**.



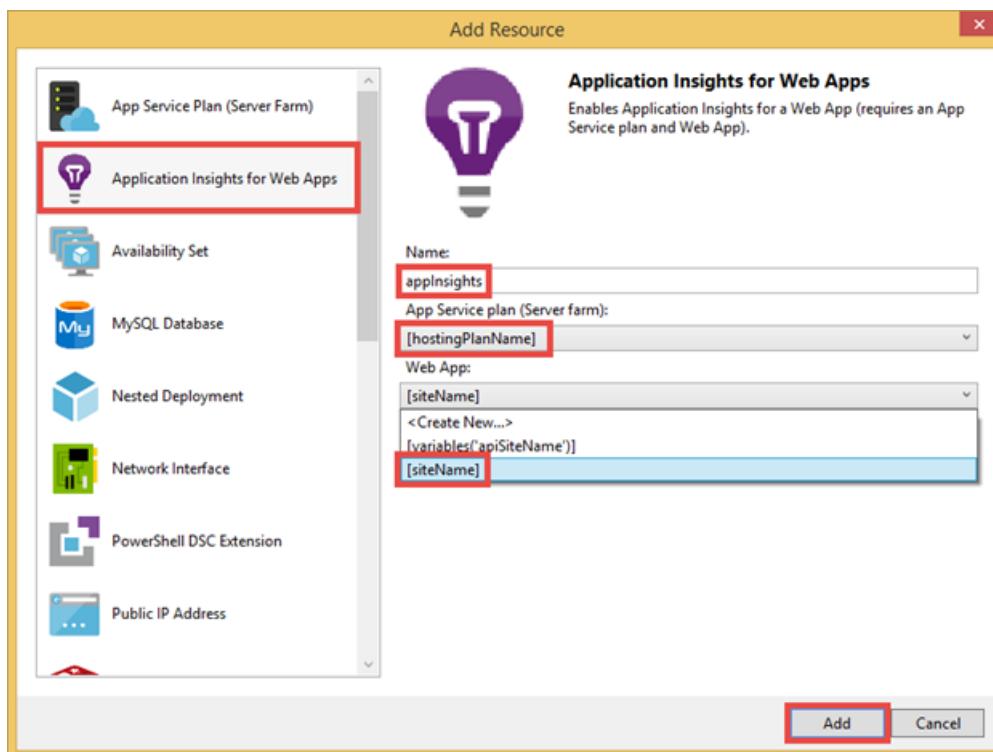
3. In **Select Azure Template**, select **Blank Template** and click **OK**.
4. Drag `azuredeploy.json` into the **Template** folder of your new project.



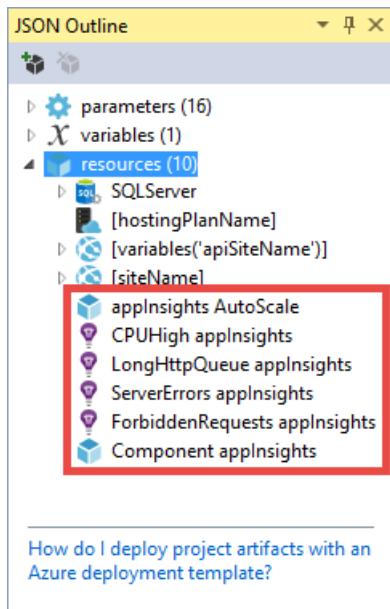
5. From Solution Explorer, open the copied `azuredeploy.json`.
6. Just for the sake of the demonstration, let's add some standard Application Insight resources to our JSON file, by clicking **Add Resource**. If you're just interested in deploying the JSON file, skip to the deploy steps.



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 **applInsights 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.

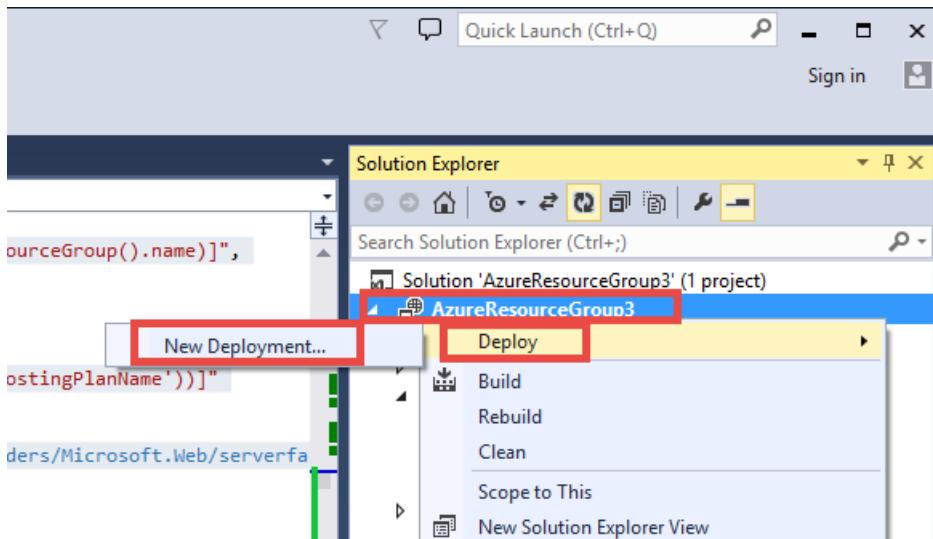
```
{
  "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
  "type": "Microsoft.Insights/autoscalesettings",
  "location": "[parameters('siteLocation')]",
  "apiVersion": "2014-04-01",
  "dependsOn": [...],
  "tags": [...],
  "properties": {
    "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
    "profiles": [...],
    "enabled": true,
    "targetResourceUri": "[concat(resourceGroup().id, '/providers/Microsoft.W
  },
}
```

10. In the JSON Outline, click **CPUHigh applInsights** to highlight its JSON code. This is an alert.

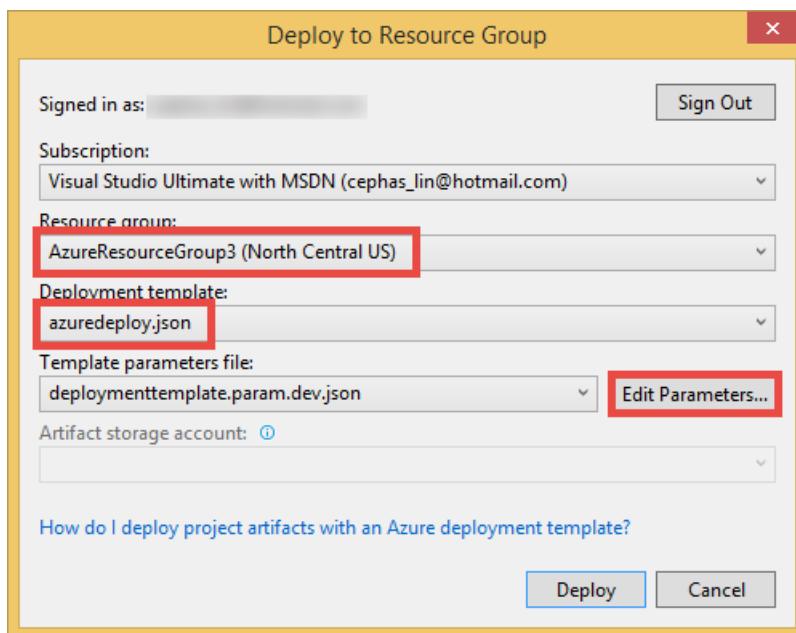
11. Locate the `location` and `isEnabled` properties and set them as shown below. Do the same for the other three alerts (purple bulbs).

```
{
  "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": ...
  }
},
```

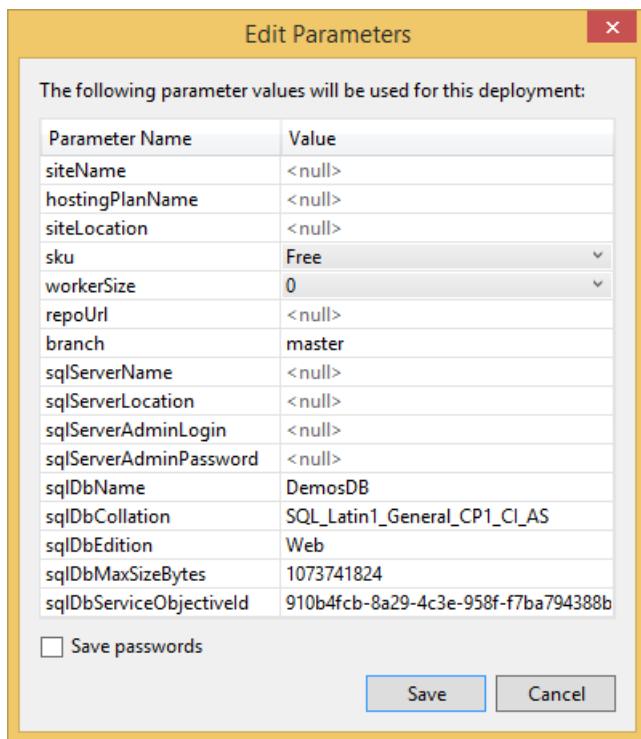
12. You're now ready to deploy. Right-click the project and select **Deploy > New Deployment**.



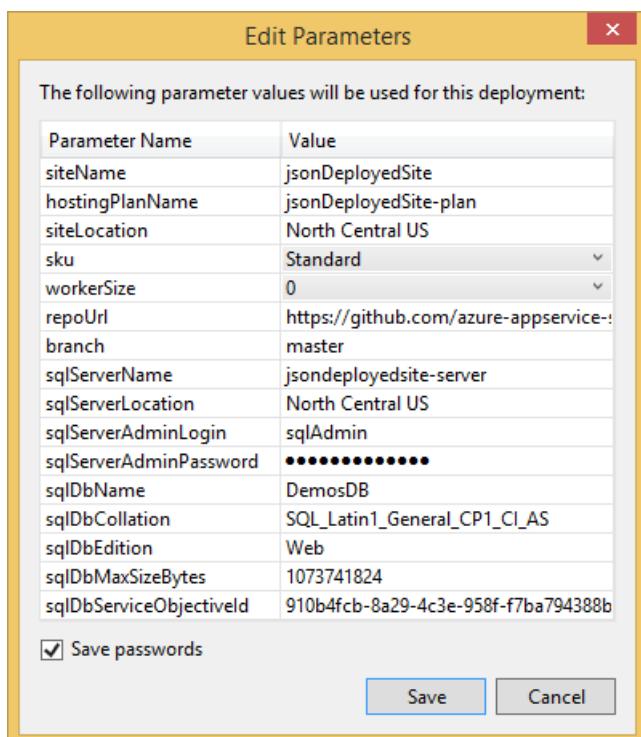
13. Log into your Azure account if you haven't already done so.
14. Select an existing resource group in your subscription or create a new one, select **azuredeploy.json**, and then click **Edit Parameters**.



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))",
        "Client-Subtype", "WindowsPowerShell"
    )
    $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.

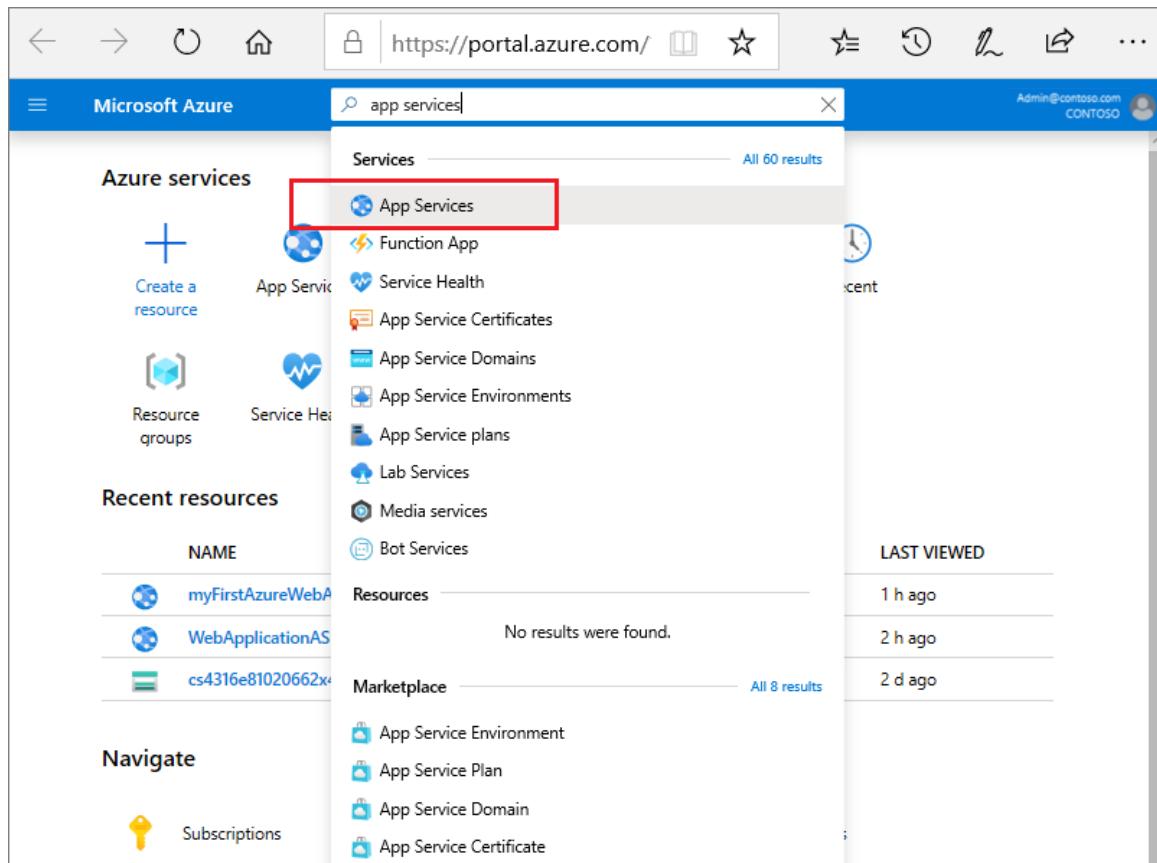
Each App Service plan tier supports a different number of deployment slots. There's no additional charge for using deployment slots. To find out the number of slots your app's tier supports, see [App Service limits](#).

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.



2. In the left pane, select **Deployment slots > Add Slot**.

A screenshot of the "my-demo-app - Deployment slots" blade in the Azure portal. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings, and Configuration. The "Deployment slots" section is currently selected. It displays a message: "You haven't added any deployment slots. Click here to get started." Below this is a "Deployment Slots" section with a table:

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app	Running	myAppServicePlan	100

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.

Add a slot

Name
staging

Clone settings from:
Do not clone settings

Add Close

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.

my-demo-app - Deployment slots

Deployment Slots

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.

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app	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.

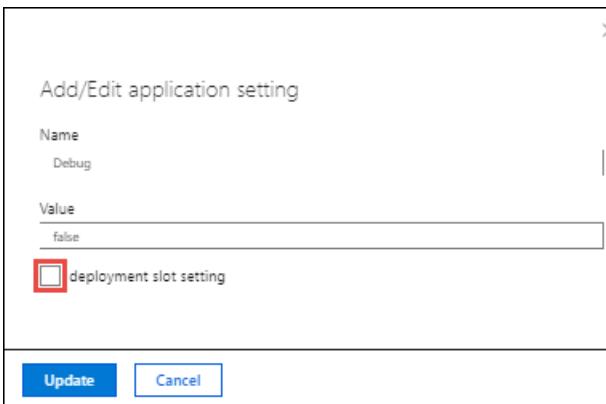
Settings that aren't swapped:

- 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 **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**.

SETTING	TYPE	OLD VALUE	NEW VALUE
MyDbConnection	ConnectionString	Server=tcp:stagingser... Info=False;User ID=<username>;Password=<password>;Multiple... Timeout=30;	Server=tcp:productio... Info=False;User ID=<username>;Password=<password>;Multiple... Timeout=30;

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)

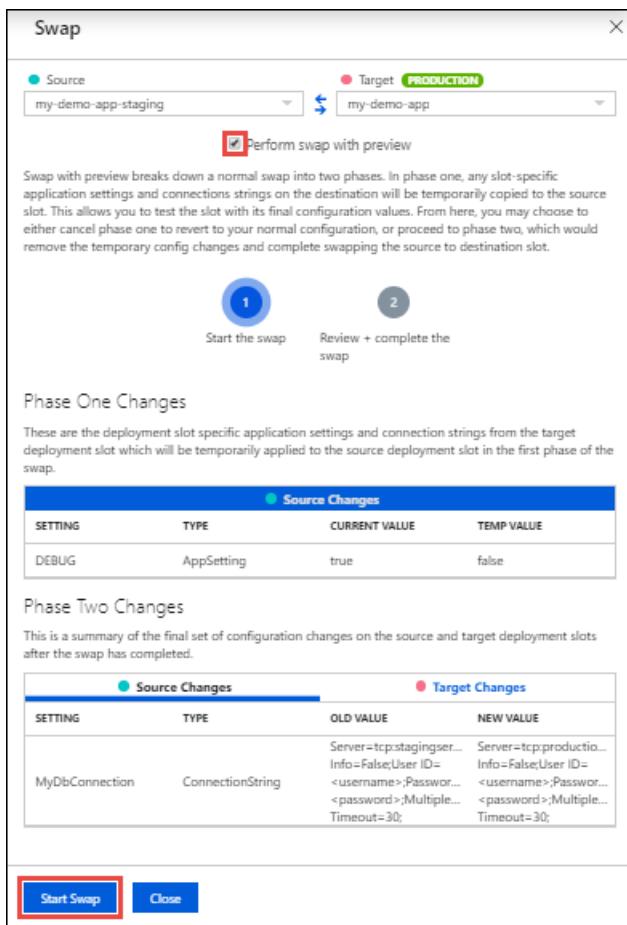
Before you swap into production as the target slot, validate that the app runs with the swapped settings. The source slot is also warmed up before the swap completion, which is desirable for mission-critical applications.

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.

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

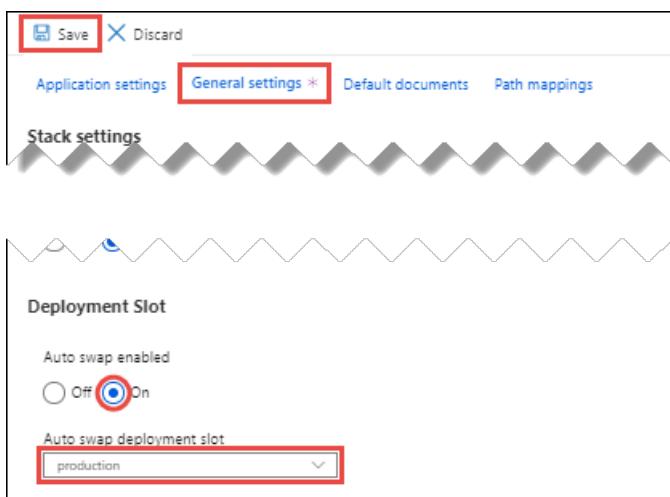
cold starts and zero downtime for customers of the app. When auto swap is enabled from a slot into production, every time you push your code changes to that slot, App Service automatically [swaps the app into production](#) after it's warmed up in the source slot.

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.

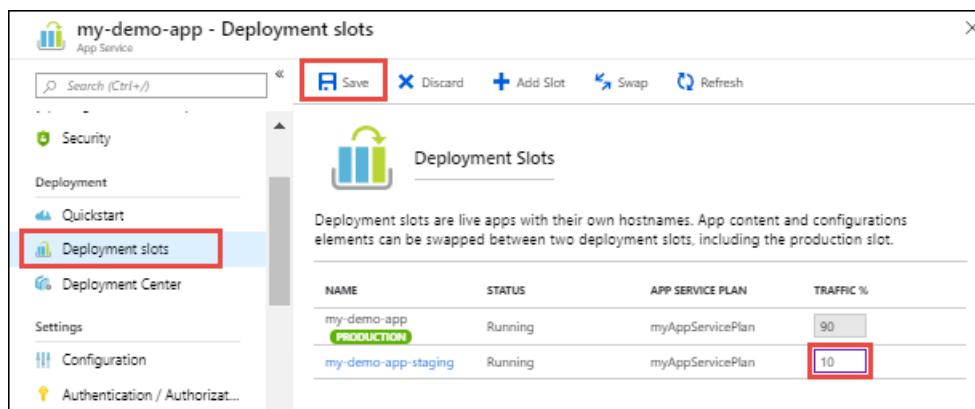
Route traffic

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	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.

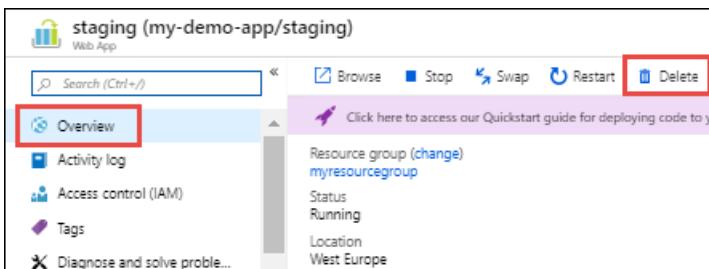
To let users opt in to your beta app, set the same query parameter to the name of the non-production slot. Here's an example:

```
<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](#)

Guidance on deploying web apps by using Azure Resource Manager templates

2/14/2020 • 3 minutes to read • [Edit Online](#)

This article provides recommendations for creating Azure Resource Manager templates to deploy Azure App Service solutions. These recommendations can help you avoid common problems.

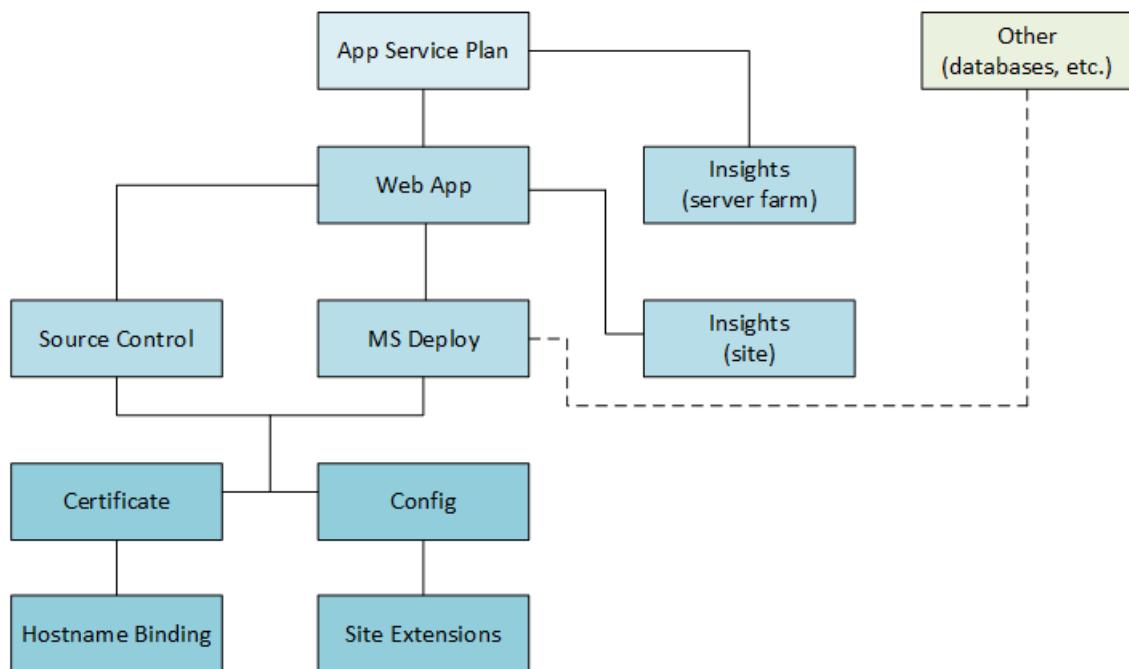
Define dependencies

Defining dependencies for web apps requires an understanding of how the resources within a web app interact. If you specify dependencies in an incorrect order, you might cause deployment errors or create a race condition that stalls the deployment.

WARNING

If you include an MSDeploy site extension in your template, you must set any configuration resources as dependent on the MSDeploy resource. Configuration changes cause the site to restart asynchronously. By making the configuration resources dependent on MSDeploy, you ensure that MSDeploy finishes before the site restarts. Without these dependencies, the site might restart during the deployment process of MSDeploy. For an example template, see [WordPress Template with Web Deploy Dependency](#).

The following image shows the dependency order for various App Service resources:



You deploy resources in the following order:

Tier 1

- App Service plan.
- Any other related resources, like databases or storage accounts.

Tier 2

- Web app--depends on the App Service plan.

- Azure Application Insights instance that targets the server farm--depends on the App Service plan.

Tier 3

- Source control--depends on the web app.
- MSDeploy site extension--depends on the web app.
- Azure Application Insights instance that targets the web app--depends on the web app.

Tier 4

- App Service certificate--depends on source control or MSDeploy if either is present. Otherwise, it depends on the web app.
- Configuration settings (connection strings, web.config values, app settings)--depends on source control or MSDeploy if either is present. Otherwise, it depends on the web app.

Tier 5

- Host name bindings--depends on the certificate if present. Otherwise, it depends on a higher-level resource.
- Site extensions--depends on configuration settings if present. Otherwise, it depends on a higher-level resource.

Typically, your solution includes only some of these resources and tiers. For missing tiers, map lower resources to the next-higher tier.

The following example shows part of a template. The value of the connection string configuration depends on the MSDeploy extension. The MSDeploy extension depends on the web app and database.

```
{
  "name": "[parameters('appName')]",
  "type": "Microsoft.Web/Sites",
  ...
  "resources": [
    {
      "name": "MSDeploy",
      "type": "Extensions",
      "dependsOn": [
        "[concat('Microsoft.Web/Sites/', parameters('appName'))]",
        "[concat('Microsoft.Sql/servers/', parameters('dbServerName'), '/databases/',
        parameters('dbName'))]",
      ],
      ...
    },
    {
      "name": "connectionstrings",
      "type": "config",
      "dependsOn": [
        "[concat('Microsoft.Web/Sites/', parameters('appName'), '/Extensions/MSDeploy')]"
      ],
      ...
    }
  ]
}
```

For a ready-to-run sample that uses the code above, see [Template: Build a simple Umbraco Web App](#).

Find information about MSDeploy errors

If your Resource Manager template uses MSDeploy, the deployment error messages can be difficult to understand. To get more information after a failed deployment, try the following steps:

1. Go to the site's [Kudu console](#).
2. Browse to the folder at D:\home\LogFiles\SiteExtensions\MSDeploy.

3. Look for the appManagerStatus.xml and appManagerLog.xml files. The first file logs the status. The second file logs information about the error. If the error isn't clear to you, you can include it when you're asking for help on the [forum](#).

Choose a unique web app name

The name for your web app must be globally unique. You can use a naming convention that's likely to be unique, or you can use the [uniqueString function](#) to assist with generating a unique name.

```
{  
  "apiVersion": "2016-08-01",  
  "name": "[concat(parameters('siteNamePrefix'), uniqueString(resourceGroup().id))]",  
  "type": "Microsoft.Web/sites",  
  ...  
}
```

Deploy web app certificate from Key Vault

NOTE

This article has been updated to use the 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](#).

If your template includes a [Microsoft.Web/certificates](#) resource for SSL binding, and the certificate is stored in a Key Vault, you must make sure the App Service identity can access the certificate.

In global Azure, the App Service service principal has the ID of **abfa0a7c-a6b6-4736-8310-5855508787cd**. To grant access to Key Vault for the App Service service principal, use:

```
Set-AzKeyVaultAccessPolicy `  
  -VaultName KEY_VAULT_NAME `  
  -ServicePrincipalName abfa0a7c-a6b6-4736-8310-5855508787cd `  
  -PermissionsToSecrets get `  
  -PermissionsToCertificates get
```

In Azure Government, the App Service service principal has the ID of **6a02c803-daf3-4136-b4c3-5a6f318b4714**. Use that ID in the preceding example.

In your Key Vault, select **Certificates** and **Generate/Import** to upload the certificate.

The screenshot shows the 'Certificates' blade in the Azure Key Vault. The left sidebar has links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Keys, Secrets, and Certificates. The 'Certificates' link is highlighted with a red box. The main content area has a search bar, a 'Generate/Import' button (also highlighted with a red box), and a message: 'NAME There are no certificates available.'

In your template, provide the name of the certificate for the `keyVaultSecretName`.

For an example template, see [Deploy a Web App certificate from Key Vault secret and use it for creating SSL binding](#).

Next steps

- For a tutorial on deploying web apps with a template, see [Provision and deploy microservices predictably in Azure](#).
- To learn about JSON syntax and properties for resource types in templates, see [Azure Resource Manager template reference](#).

Buy a custom domain name for Azure App Service

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

<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**

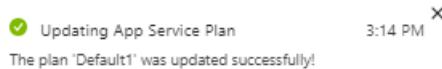
Memory per instance available to run applications deployed and running in...

**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.



Buy the domain

Pricing Information

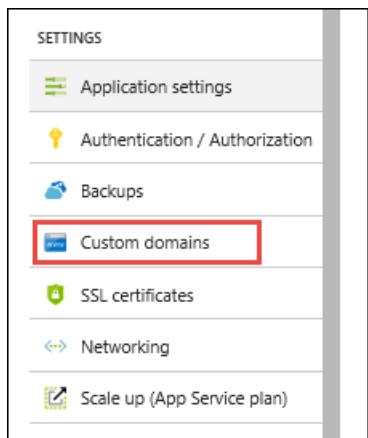
For pricing information on Azure App Service Domains, visit the [App Service Pricing page](#) and scroll down to App Service Domain.

[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**.

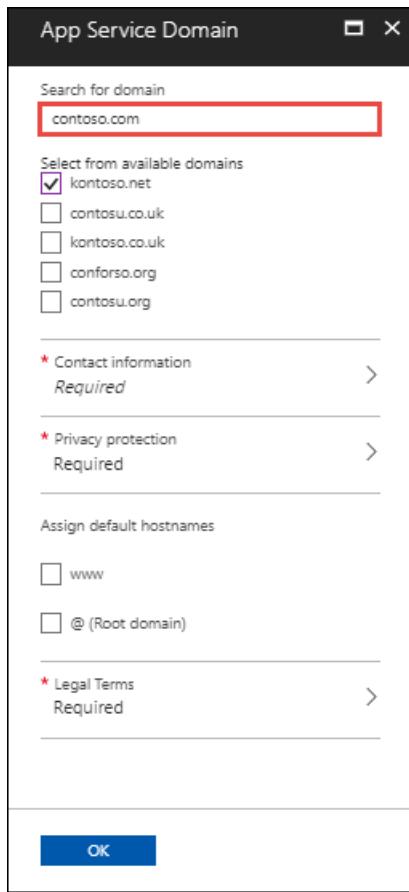
A screenshot of the 'App Service Domains' page. At the top left is a 'WWW' icon. To its right is the title 'App Service Domains'. Below the title is a brief description: 'Purchase and manage domains for your Azure services with auto-renew and privacy protection.' followed by a 'Learn more' link. In the center of the page is a large button with a plus sign and the text 'Buy Domain', which is highlighted with a red box. Below this 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 Privacy protection page.
Assign default hostnames	www and @	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 Azure Traffic Manager , 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.

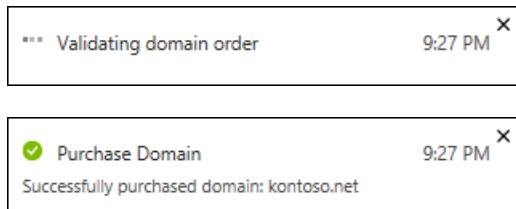
Accept terms and purchase

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. There are input fields for 'IP address' (13.69.68.6) and 'HTTPS Only' (set to 'Off'). A 'Status Filter' dropdown shows 'All (3)' selected. The main table has columns for 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. The data rows are:

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

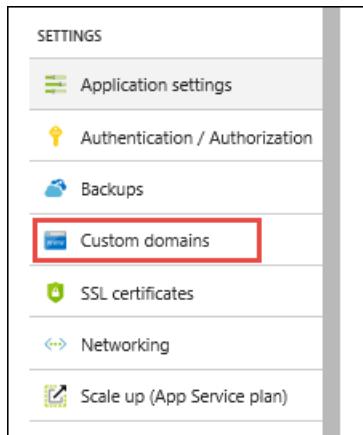
If you choose not to assign one or more default hostnames to your app during the purchase process, or if you need to assign a hostname not listed, you can assign a hostname anytime.

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

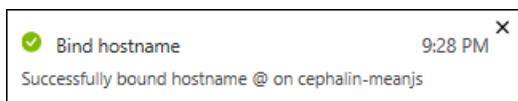
In the **Add hostname** dialog, type the fully qualified domain name of your App Service Domain or any subdomain. For example:

- 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

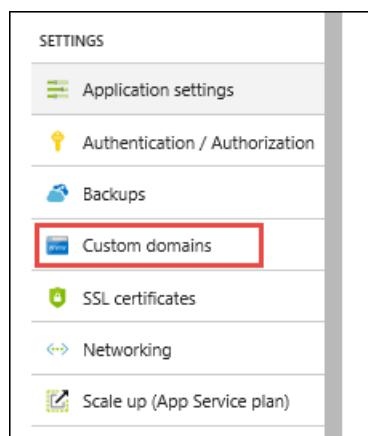
Navigate to the listed hostnames in the browser. In the example in the preceding screenshot, try navigating to *abc.kontoso.net*.

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 sidebar lists various management options 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 un-expected domain ownership loss.' Below this is a switch labeled 'Auto renew domain:' with the 'Off' option 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 "app service". Below the search bar, the results are displayed under the heading "All services". The results include:

- App Service Certificates
- App Service Domains
- App Service Environments
- App Service plans
- App Services
Keywords: web apps
- Citrix XenDesktop Essentials
Keywords: VDI appliance
- Cloud services (classic)
Keywords: apps
- Service catalog managed application definitions

A blue star icon is present next to each result item.

Select the domain to manage.

Access DNS zone

In the domain's left menu, select **DNS zone**.

The screenshot shows the Azure portal's configuration page for an App Service Domain named "kontoso.net". The left sidebar lists several management options: Automation script, Hostname bindings, Domain renewal, and DNS zone. The "DNS zone" option is highlighted with a red border. The main pane displays the "Essentials" section with the following details:

- Resource group: myResourceGroup
- Status: Active
- Location: global
- Subscription name: (redacted)
- Subscription ID: (redacted)

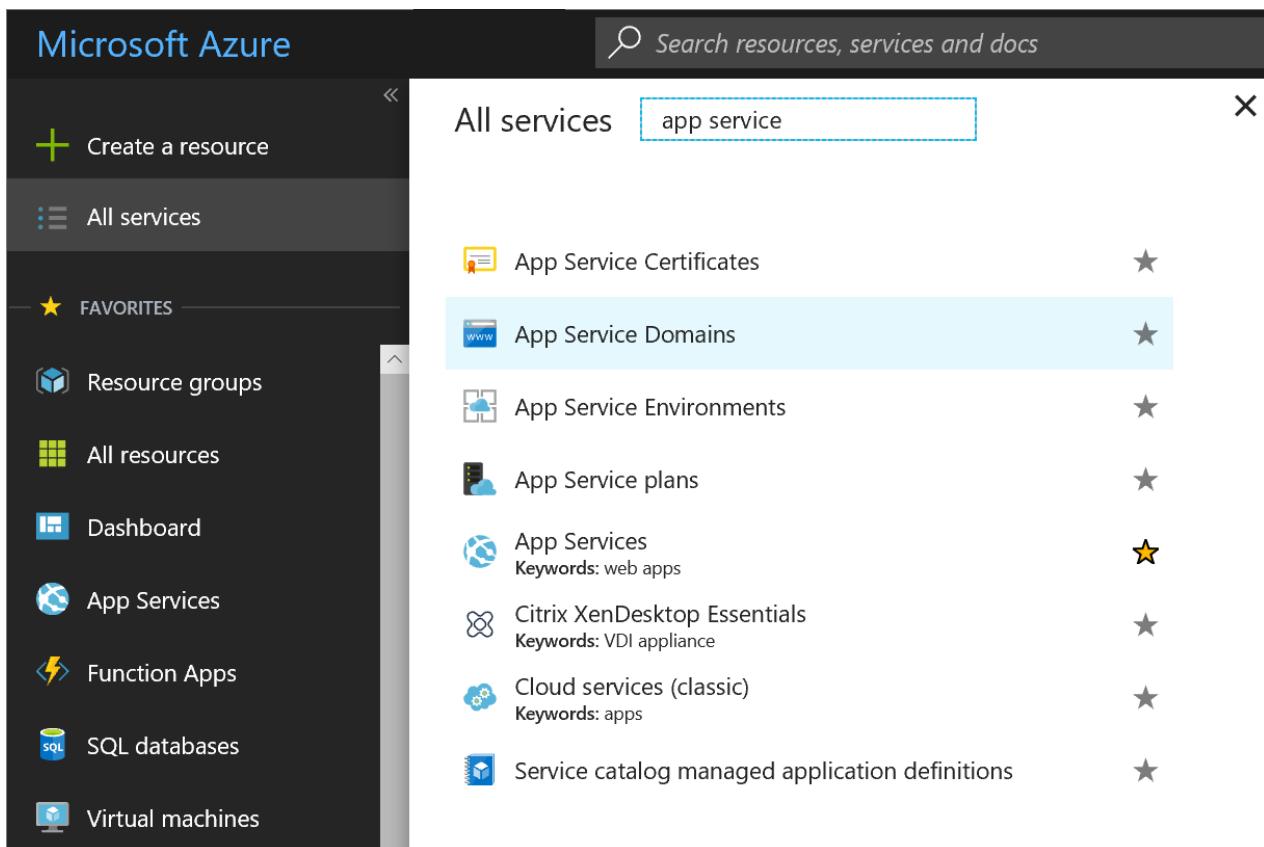
This action opens the [DNS zone](#) page of your App Service Domain in Azure DNS. For information on how to edit DNS records, see [How to manage DNS Zones in the Azure portal](#).

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**.

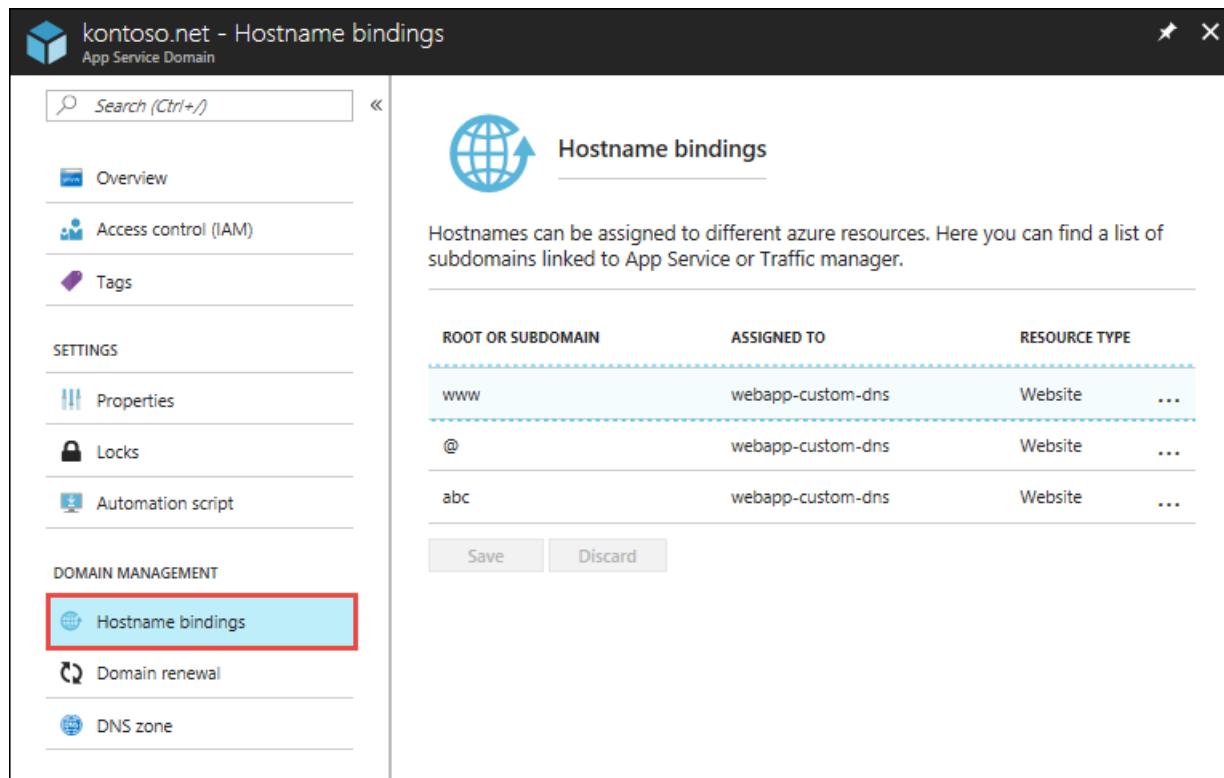


The screenshot shows the Microsoft Azure search interface. The search bar at the top contains the text "Search resources, services and docs". Below the search bar, the text "All services" is displayed, followed by a search input field containing "app service". A list of services is shown, each with a 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

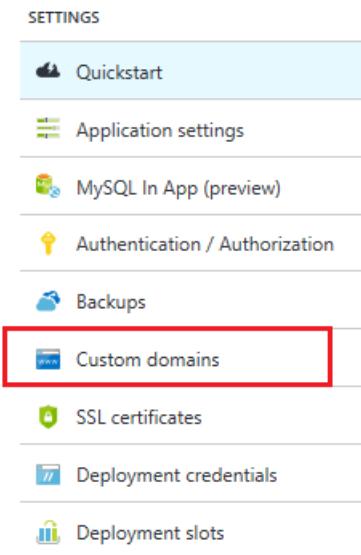
It can take some time for your CNAME to propagate through the DNS system. You can use a service such as <https://www.digwebinterface.com/> to verify that the CNAME is available.

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. On the left, there are sections for Purchased Domains (with a 'Buy Domain' button) and Hostnames (with an 'Add hostname' button, which is highlighted with a red box). On the right, there is a form for adding a hostname:

- Hostname:** www.contoso.com (with a green checkmark icon)
- Validate** button (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><appname>.azurewebsites.net</i>
www (sub)	<i>awverify.www</i>	<i><appname>.azurewebsites.net</i>
* (wildcard)	<i>awverify.*</i>	<i><appname>.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

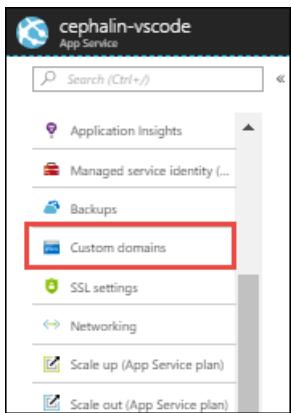
For certain providers, such as CloudFlare, `awverify.*` is not a valid record. Use `*` only instead.

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 '13.0.9.68.6' and 'HTTPS Only:' with a toggle switch set to 'Off'. A large blue '+' button is highlighted with a red box, labeled 'Add custom domain'. Below this is a 'Status Filter' section with three tabs: 'All (1)' (highlighted with a red box), 'Not Secure (0)', and 'Secure (1)'. Underneath is a table with columns 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. One row shows a green checkmark under 'SSL STATE', the domain 'my-demo-app.azurewebsites.net' under 'ASSIGNED CUSTOM DOMAINS', and an empty column under 'SSL BINDING'.

Type the fully qualified domain name that you added the TXT record for, such as `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 a dropdown 'my-demo-app'. The first field, 'Custom domain', contains 'www.contoso.com' and is highlighted with a red box. Below it is a blue placeholder box. The 'Hostname record type' dropdown is set to 'CNAME (www.example.com or any subdomain)' and is also highlighted with a red box. A section titled 'CNAME configuration' explains what a CNAME record is, and below it is a 'CNAME' input field containing 'my-demo-app.azurewebsites.net'. At the bottom is a large blue 'Add custom domain' button.

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 for 'Not Secure' (www.contoso.com) which has an 'Add binding' button and a '...' menu item; and one for 'Secure' (my-demo-app.azurewebsites.net) which is marked with a green checkmark.

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. At the top, there's a status bar with 'IP address: 40.118.102.46' and an 'HTTPS Only' toggle switch set to 'Off'. Below this is a button to 'Add hostname'. The main table has columns for 'HOSTNAMES ASSIGNED TO SITE' and 'SSL BINDING'. It lists one entry: cephalin-vscode.azurewebsites.net.

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 Copy the app's IP address
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 www custom domain 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 Azure Key Vault to manage your PKCS12 certificates . See Private certificate requirements .
Upload a private certificate	If you already have a private certificate from a third-party provider, you can upload it. See Private certificate requirements .
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

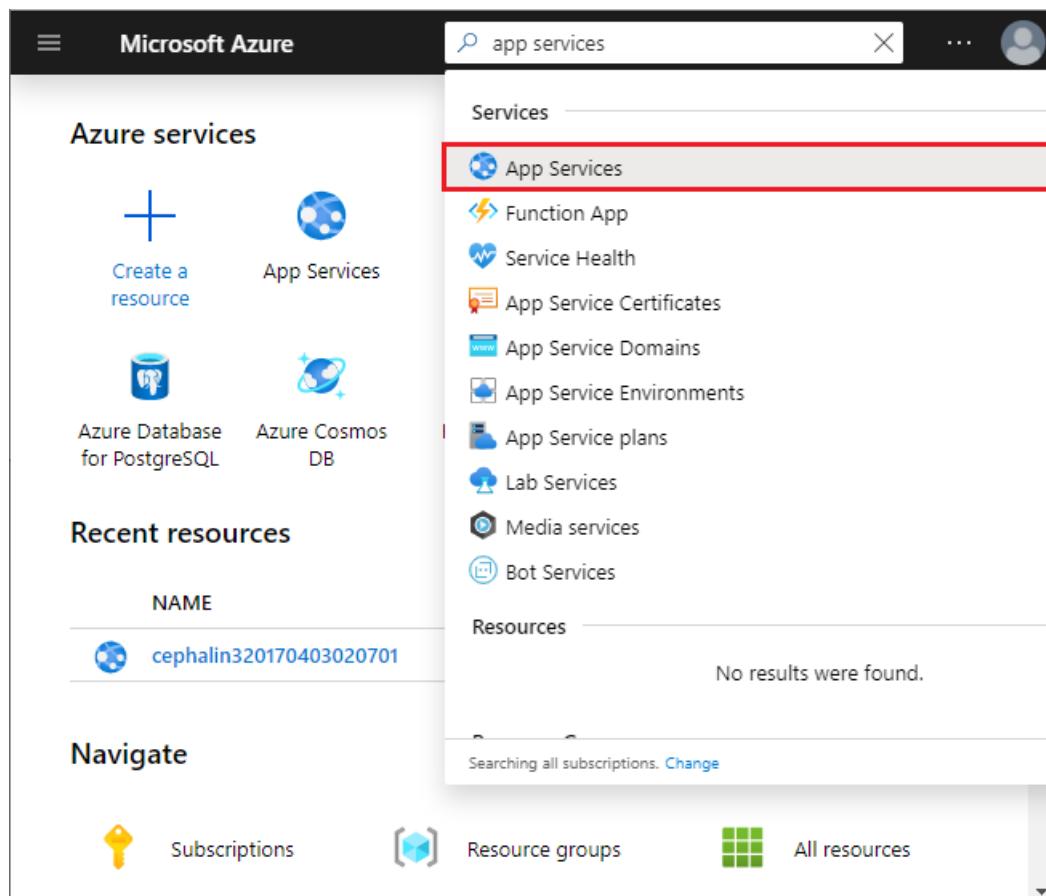
To bind a custom SSL certificate (a third-party certificate or App Service certificate) to your web app, your [App Service plan](#) must be in the **Basic, Standard, Premium**, or **Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

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 'app services' typed into it. Below the search bar, there's a sidebar titled 'Azure services' with icons for 'Create a resource', 'App Services', 'Azure Database for PostgreSQL', and 'Azure Cosmos DB'. To the right of the sidebar, the main content area is titled 'Services' and lists several options: 'App Services' (which is 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'. Below this list, there's a section titled 'Resources' with the message 'No results were found.' At the bottom of the page, 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 name 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

Shared infrastructure

1 GB memory

60 minutes/day compute

Free

F1

Shared infrastructure

1 GB memory

240 minutes/day compute

<price>/Month (Estimated)

D1

100 total ACU

1.75 GB memory

A-Series compute equivalent

<price>/Month (Estimated)

B1

▼ See additional options

Included features

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



Custom domains / SSL

Configure and purchase custom domains with SNI SSL bindings



Manual scale

Up to 3 instances. Subject to availability.

Included hardware

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



Azure Compute Units (ACU)

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



Memory

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



Storage

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

Apply

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

Updating App Service Plan

3:14 PM

The plan 'Default1' was updated successfully!

Create a free 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 visible. At the bottom of this section, there are four buttons: '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 overall layout includes a search bar at the top and various navigation links like 'Quickstart', 'Deployment slots', and 'Deployment Center' on the left sidebar.

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' at the top is set to 'All'. The table below lists one certificate entry:

Health Status	Hostname	Expiration	Thumbprint	...
✓ Healthy	www.contoso.com	4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

Import an App Service Certificate

If you purchase an App Service Certificate from Azure, Azure manages the following tasks:

- Takes care of the purchase process from GoDaddy.
- Performs domain verification of the certificate.
- Maintains the certificate in [Azure Key Vault](#).
- Manages certificate renewal (see [Renew certificate](#)).
- Synchronizes the certificate automatically with the imported copies in App Service apps.

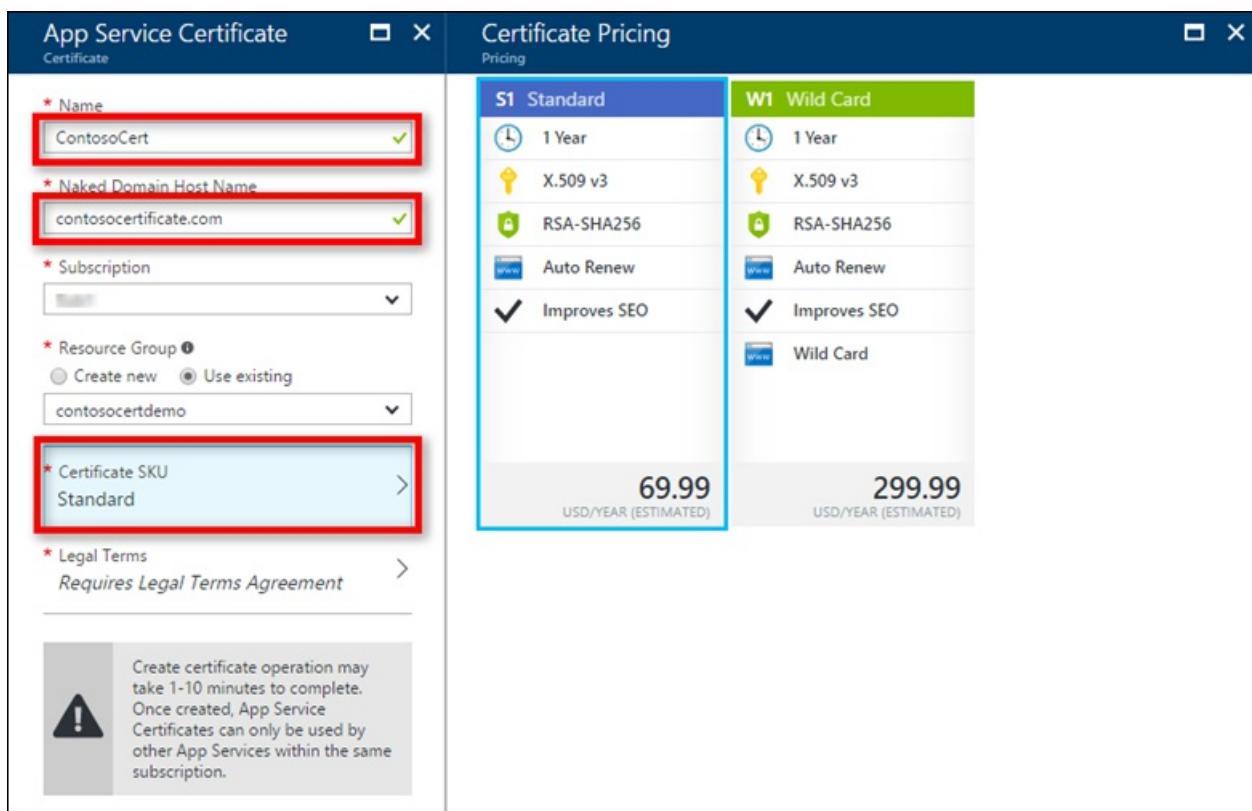
To purchase an App Service certificate, go to [Start certificate order](#).

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](#).



Use the following table to help you configure the certificate. When finished, click **Create**.

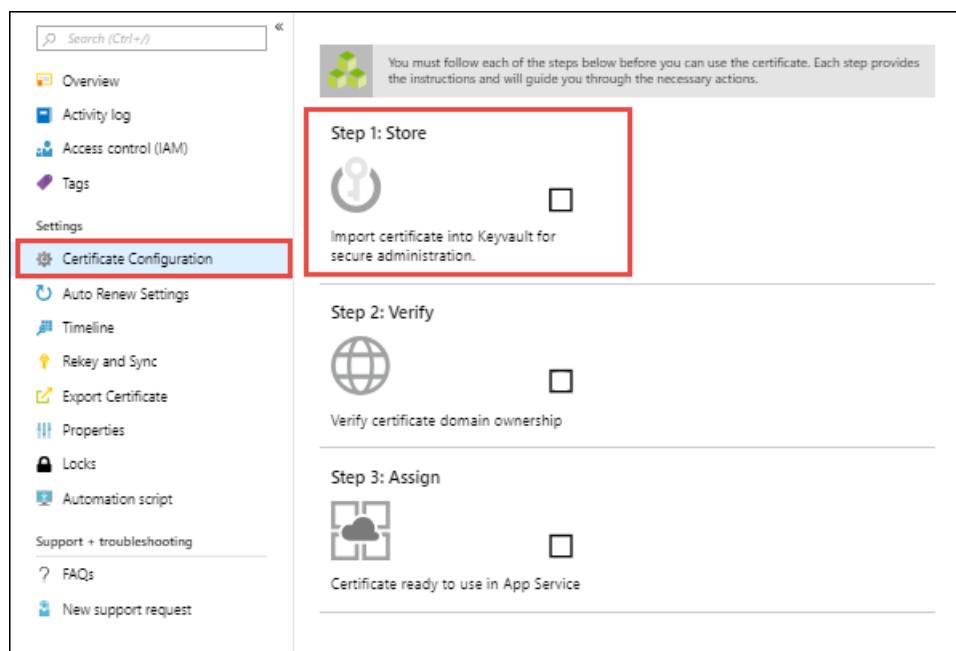
SETTING	DESCRIPTION
Name	A friendly name for your App Service certificate.
Naked Domain Host Name	Specify the root domain here. The issued certificate secures <i>both</i> the root domain and the <code>www</code> subdomain. In the issued certificate, the Common Name field contains the root domain, and 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 wildcard certificate .
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 Azure Key Vault pricing details .

SETTING	DESCRIPTION
Access policies	Defines the applications and the allowed access to the vault resources. You can configure it later, following the steps at Grant several applications access to a key vault .
Virtual Network Access	Restrict vault access to certain Azure virtual networks. You can configure it later, following the steps at Configure Azure Key Vault Firewalls and Virtual Networks

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 'Certificate Configuration' page for a key vault. On the left is a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Settings, Certificate Configuration (which is selected), Auto Renew Settings, Timeline, Rekey and Sync, Export Certificate, Properties, Locks, Automation script, Support + troubleshooting, FAQs, and New support request. The main area has a heading 'You must follow each of the steps below before you can use the certificate. Each step provides the instructions and will guide you through the necessary actions.' Below this, 'Step 1: Store' is shown with a green checkmark and the message 'Certificate successfully imported to Key vault.' 'Step 2: Verify' is highlighted with a red box and contains a globe icon and a checkbox labeled 'Verify certificate domain ownership'. 'Step 3: Assign' follows with a cloud icon and a checkbox labeled '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 Management Portal interface. On the left, there's a navigation menu with various options like 'Quickstart', 'Deployment slots', 'Deployment Center', 'Settings' (which is expanded), 'TLS/SSL settings' (highlighted with a red box), 'Networking', 'Scale up (App Service plan)', and 'Scale out (App Service plan)'. The main content area has tabs for 'Bindings', 'Private Key Certificates (.pfx)' (which is selected and highlighted with a red box), and 'Public Key Certificates (.cer)'. Below these tabs, there's a section titled 'Private Key Certificate' with a 'PFX' icon. A note says: '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	...
Healthy	contoso.com, www.contoso.com	10/10/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

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. The table below shows one row of data:

Health Status	Hostname	Expiration	Thumbprint	...
Healthy	contoso.com, www.contoso.com	10/10/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

Import a certificate from Key Vault

If you use Azure Key Vault to manage your certificates, you can import a PKCS12 certificate from Key Vault into App Service as long as it [satisfies the requirements](#).

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**.

The screenshot shows the Azure Management Portal interface. On the left, there's a sidebar with various settings like Configuration, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, and TLS/SSL settings. The 'TLS/SSL settings' option is highlighted with a red box. The main area shows 'Bindings' with tabs for 'Private Key Certificates (.pfx)', 'Public Key Certificates (.cer)', and 'PFX'. Under 'Private Key Certificates (.pfx)', there's a sub-section for 'Private Key Certificate'. It contains a note about using .pfx files for TLS/SSL bindings and instructions to learn more or upload certificates. Below this are four buttons: 'Import App Service Certificate', 'Upload Certificate', 'Import Key Vault Certificate' (which is also highlighted with a red box), and 'Create App Service Managed Certificate'. A table titled 'Private Key Certificates' shows a status filter with 'All' selected, and columns for Health St..., Hostname, Expiration, and Thumbprint. The table displays the message '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](#).

The screenshot shows the 'Private Key Certificates' list. At the top, there's a 'Status Filter' with 'All' selected. Below it is a table with columns for 'Health Status', 'Hostname', 'Expiration', and 'Thumbprint'. One row is visible, showing a green checkmark in the 'Healthy' column, the hostname 'contoso.com, www.contoso.com', the expiration date '10/10/2020', the thumbprint '6A3BCCA5CC4B0158F0A097CE9F39...', and three vertical dots for more options.

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**.

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	...
Healthy	www.contoso.com	4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

In **PFX Certificate File**, select your PFX file. In **Certificate password**, type the password that you created when you exported the PFX file. When finished, click **Upload**.

When the operation completes, you see the certificate in the **Private Key Certificates** list.

Private Key Certificates

Status Filter: All, Healthy, Warning, Expired

Health Status	Hostname	Expiration	Thumbprint	...
Healthy	www.contoso.com	4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

Upload a public certificate

Public certificates are supported in the .cer format.

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, click **TLS/SSL settings > Public Certificates (.cer) > Upload Public Key Certificate**.

In **Name**, type a name for the certificate. In **CER Certificate file**, select your CER file.

Click **Upload**.

Add Public Key Certificate (.cer) 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 Browse

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.

The screenshot shows the Azure portal interface for managing certificates. On the left, there's a navigation menu with items like Overview, Activity log, Access control (IAM), Tags, Certificate Configuration, Auto Renew Settings, Timeline, Rekey and Sync (which is highlighted with a red box), Export Certificate, Properties, Locks, Automation script, Support + troubleshooting, FAQs, and New support request. At the top, there are buttons for Refresh, Rekey (highlighted with a red box), Sync, and a search bar. The main content area is titled 'Rekey Certificate' and contains a key icon. It explains that rekeying will roll the certificate with a new one from the certificate authority. A note says that rekey operations are free and does not incur additional charges. Below this is a section titled 'Linked Private Certificate' with a PFX icon. It states that linked private certificates are used in App Service apps and can go out of sync after rekey and renew operation. It suggests clicking 'sync' to update the private certificates and SSL bindings. There's a table showing 'Current Certificate Thumbprint' with columns for STATUS, LINKED PRIVATE CERTIFI..., RESOURCE GROUP, and THUMBPRINT. One row is shown with a green checkmark under STATUS and 'myResourceGro...' under RESOURCE GROUP.

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.

The screenshot shows the Azure portal interface for managing certificates. The left navigation menu includes Overview, Activity log, Access control (IAM), Tags, Certificate Configuration, Auto Renew Settings (highlighted with a red box), Timeline, Rekey and Sync, Export Certificate, Properties, Locks, Automation script, Support + troubleshooting, FAQs, and Help. At the top, there are buttons for Refresh, Save (highlighted with a red box), Discard, Manual Renew, and Sync. The main content area has a warning icon and the title 'Manual renewal not allowed at this time'. It explains that the App Service Certificate is not eligible for manual renewal right now and that manual renewal will become available 60 days before expiry. It also notes that manual renewal is only for getting a new certificate with extended expiry. For rolling keys, it suggests using the Rekey feature. Below this is a section titled 'Linked Private Certificate' with a PFX icon. It states that linked private certificates are used in App Service apps and can go out of sync after rekey and renew operation. It suggests clicking 'sync' to update the private certificates and SSL bindings. There's a table showing 'Current Certificate Thumbprint' with columns for STATUS, LINKED PRIVATE CERTIFI..., RESOURCE GROUP, and THUMBPRINT. One row is shown with a green checkmark under STATUS and 'myResourceGro...' under RESOURCE GROUP.

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
```

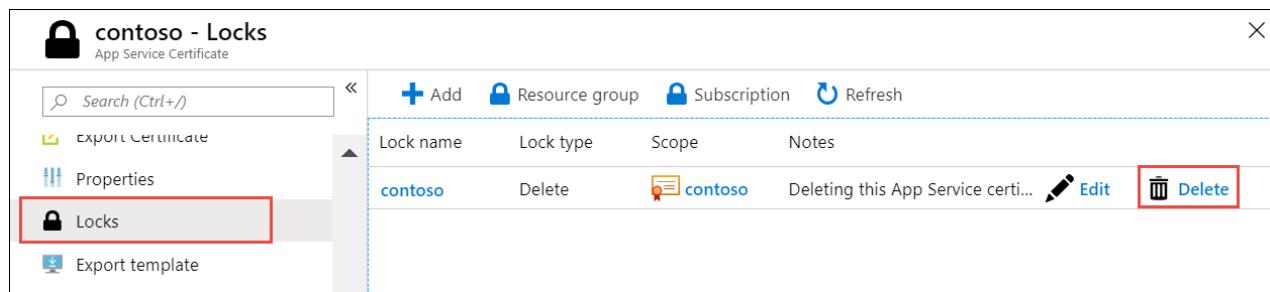
The downloaded *appservicecertificate.pfx* file is a raw PKCS12 file that contains both the public and private certificates. In each prompt, use an empty string for the import password and the PEM pass phrase.

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 certi...

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

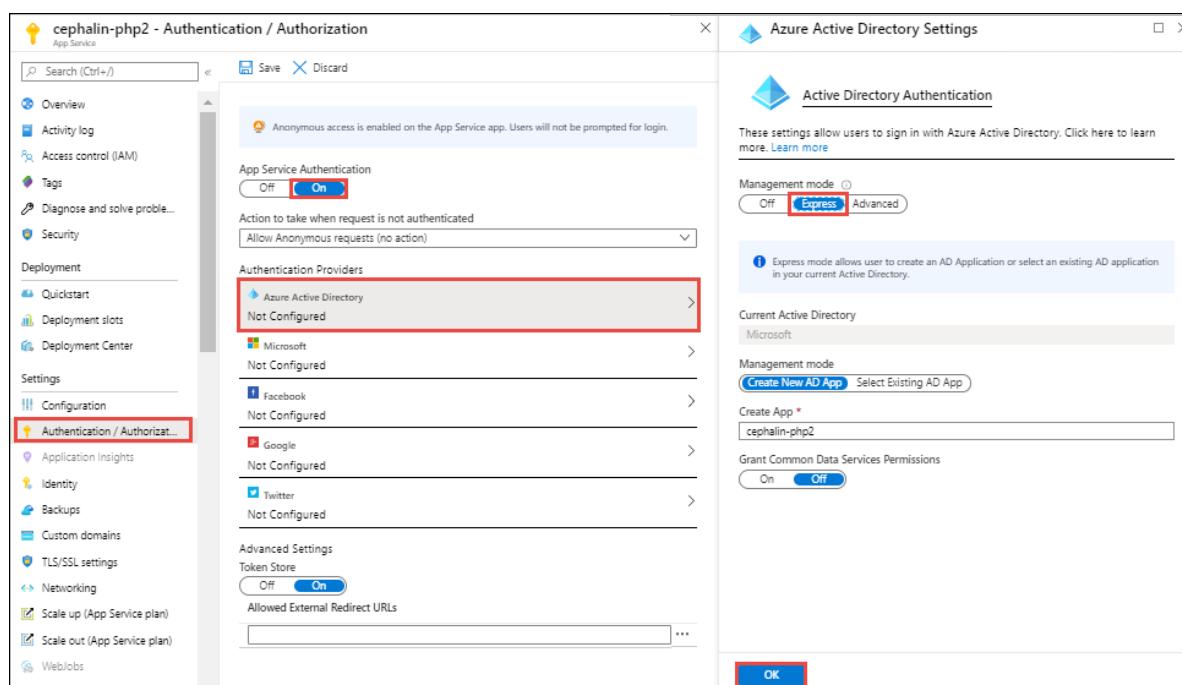
NOTE

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 Application (client) ID of the app registration.
Issuer ID	Use <code>https://login.microsoftonline.com/<tenant-id></code> , and replace <code><tenant-id></code> with the Directory (tenant) ID 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 Application ID URI of the web app here. The configured Client ID 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.

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

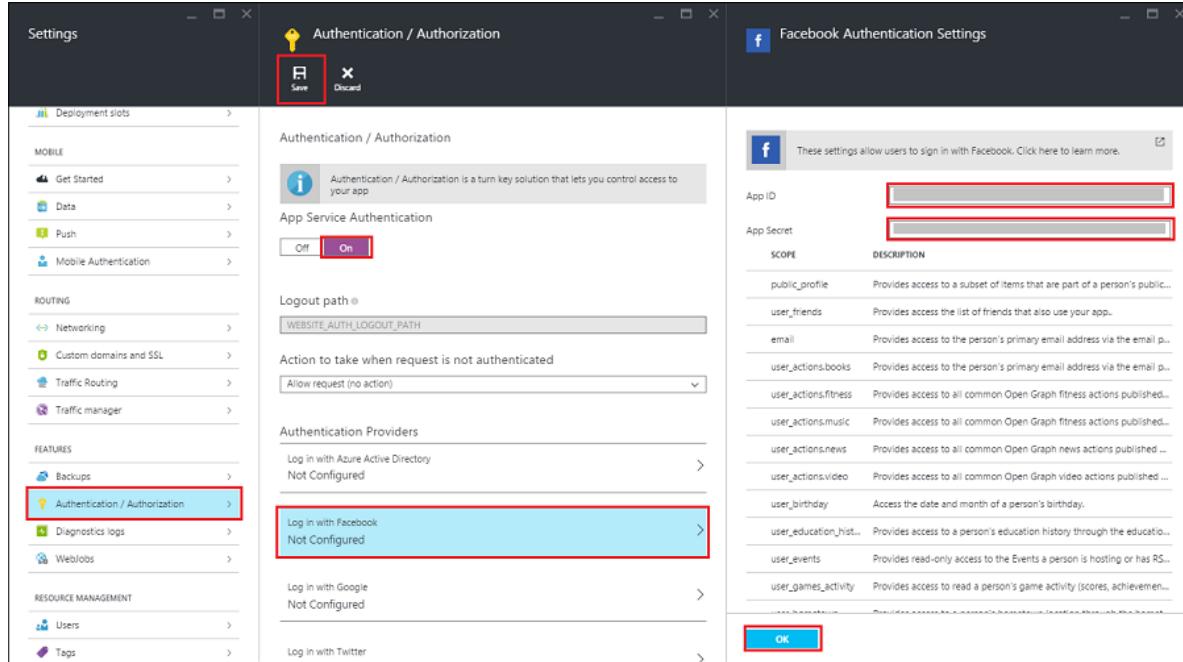
The app secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

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](#).

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:

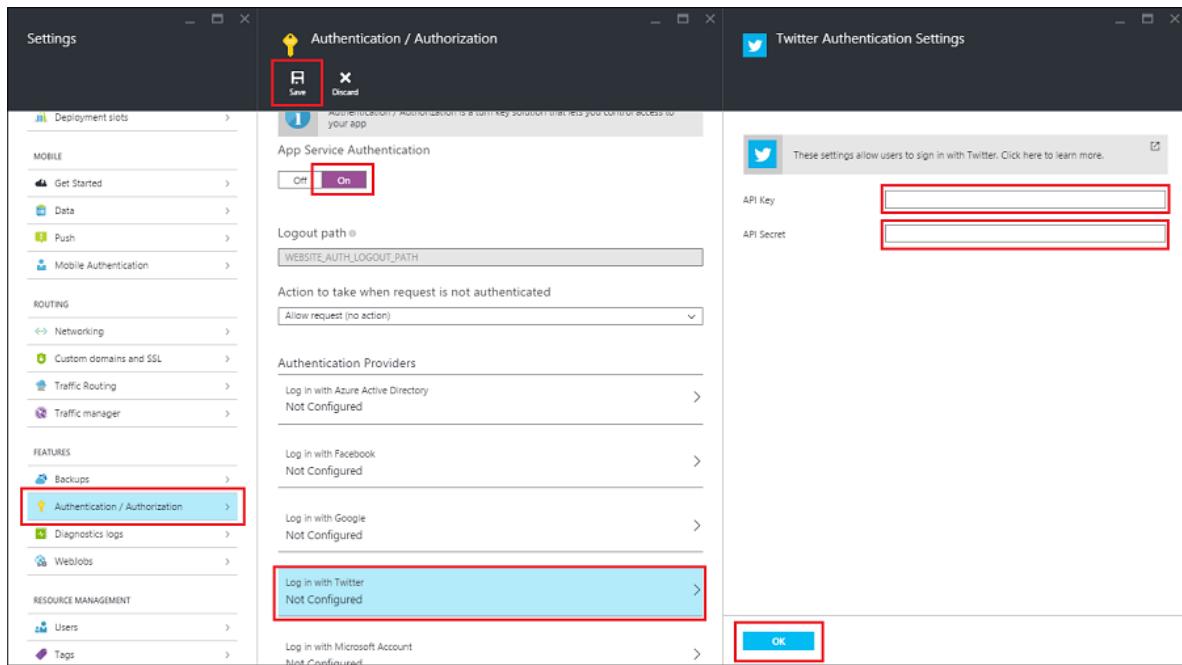
- Consumer key (API key)
- Consumer secret (API secret)

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 user access token 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 Nodejs 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 \(Linux\)](#)

Azure App Service Access Restrictions

12/20/2019 • 6 minutes to read • [Edit Online](#)

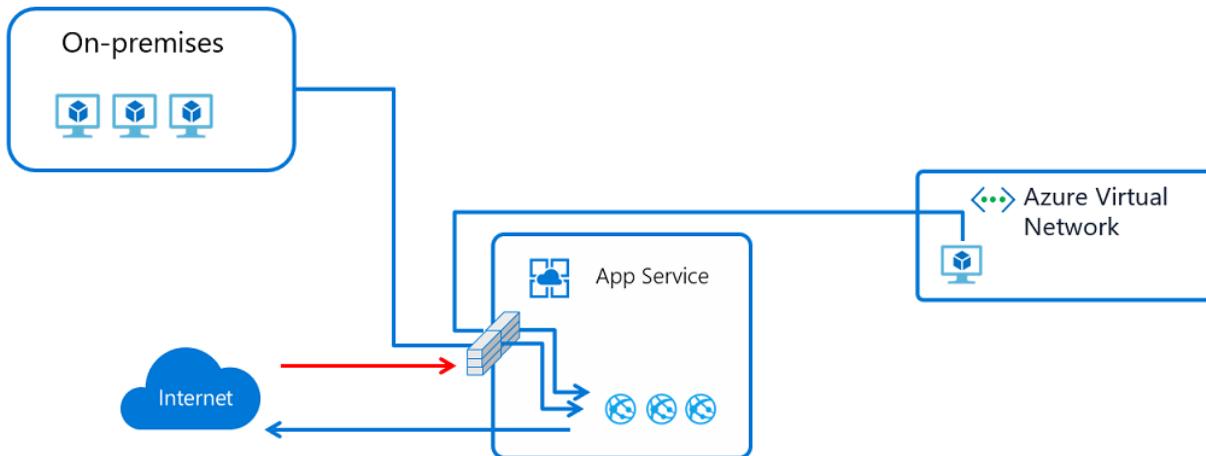
Access restrictions enable you to define a priority ordered allow/deny list that controls network access to your app. The list can include IP addresses or Azure Virtual Network subnets. When there are one or more entries, there is then an implicit "deny all" that exists at the end of the list.

The access restrictions capability works with all App Service hosted work loads including; web apps, API apps, Linux apps, Linux container apps, and Functions.

When a request is made to your app, the FROM address is evaluated against the IP address rules in your access restrictions list. If the FROM address is in a subnet that is configured with service endpoints to Microsoft.Web, then the source subnet is compared against the virtual network rules in your access restrictions list. If the address is not allowed access based on the rules in the list, the service replies with an [HTTP 403](#) status code.

The access restrictions capability is implemented in the App Service front-end roles, which are upstream of the worker hosts where your code runs. Therefore, access restrictions are effectively network ACLs.

The ability to restrict access to your web app from an Azure Virtual Network (VNet) is called [service endpoints](#). Service endpoints enable you to restrict access to a multi-tenant service from selected subnets. It must be enabled on both the networking side as well as the service that it is being enabled with. It does not work to restrict traffic to apps that are hosted in an App Service Environment. If you are in an App Service Environment, you can control access to your app with IP address rules.



Adding and editing access restriction rules in the portal

To add an access restriction rule to your app, use the menu to open **Network>Access Restrictions** and click on **Configure Access Restrictions**

The screenshot shows the Azure portal interface for managing an App Service named 'vnet-integration-app'. The left sidebar has a tree view with 'Networking' selected. The main content area displays three configuration sections: 'VNet Integration', 'Hybrid connections', and 'Azure CDN', each with a 'Click here to configure' button.

From the Access Restrictions UI, you can review the list of access restriction rules defined for your app.

The screenshot shows the 'Access Restrictions' page for the 'vnet-integration-app'. It includes a brief description of what access restrictions are and a link to learn more. Below is a table listing current rules:

PRIORITY	NAME	SOURCE	ENDPOINT STATUS	ACTION
100	IP example rule	122.133.144.0/24		Allow
150	deny example	122.133.144.32/28		Deny
200	test rule	networking-demos-vnet/simple-se-s...	Enabled	Allow
2147483647	Deny all	Any		Deny

The list will show all of the current restrictions that are on your app. If you have a VNet restriction on your app, the table will show if service endpoints are enabled for Microsoft.Web. When there are no defined restrictions on your app, your app will be accessible from anywhere.

Adding IP address rules

You can click on **[+]** **Add** to add a new access restriction rule. Once you add a rule, it will become effective immediately. Rules are enforced in priority order starting from the lowest number and going up. There is an implicit deny all that is in effect once you add even a single rule.

When creating a rule, you must select allow/deny and also the type of rule. You are also required to provide the priority value and what you are restricting access to. You can optionally add a name, and description to the rule.

Add IP Restriction

Name i

Action
Allow Deny

* Priority

Description

Type
 ▼

* Subscription
 ▼

* Virtual Network
 ▼

* Subnet
 ▼

Add rule

To set an IP address based rule, select a type of IPv4 or IPv6. IP Address notation must be specified in CIDR notation for both IPv4 and IPv6 addresses. To specify an exact address, you can use something like 1.2.3.4/32 where the first four octets represent your IP address and /32 is the mask. The IPv4 CIDR notation for all addresses is 0.0.0.0/0. To learn more about CIDR notation, you can read [Classless Inter-Domain Routing](#).

Service endpoints

Service endpoints enables you to restrict access to selected Azure virtual network subnets. To restrict access to a specific subnet, create a restriction rule with a type of Virtual Network. You can pick the subscription, VNet, and subnet you wish to allow or deny access with. If service endpoints are not already enabled with Microsoft.Web for the subnet that you selected, it will automatically be enabled for you unless you check the box asking not to do that. The situation where you would want to enable it on the app but not the subnet is largely related to if you have the permissions to enable service endpoints on the subnet or not. If you need to get somebody else to enable service endpoints on the subnet, you can check the box and have your app configured for service endpoints in anticipation of it being enabled later on the subnet.

Add IP Restriction

Name ?

Action
 Allow Deny

* Priority

Description

Type

* Subscription

* Virtual Network

* Subnet

 Selected subnet 'networking-demos-vnet/vnet-integration-subnet' does not have service endpoint enabled for Microsoft.Web. Enabling access may take up to 15 minutes to complete.

Ignore missing Microsoft.Web service endpoints

Service endpoints cannot be used to restrict access to apps that run in an App Service Environment. When your app is in an App Service Environment, you can control access to your app with IP access rules.

With service endpoints, you can configure your app with Application Gateways or other WAF devices. You can also configure multi-tier applications with secure backends. For more details on some of the possibilities, read [Networking features and App Service](#) and [Application Gateway integration with service endpoints](#).

Managing access restriction rules

You can click on any row to edit an existing access restriction rule. Edits are effective immediately including changes in priority ordering.

Edit IP Restriction

Name

* Priority

Action
 Allow Deny

Description

* Subscription

* Virtual Network

* Subnet

When you edit a rule, you cannot change the type between an IP address rule and a Virtual Network rule.

 Edit IP Restriction X

Name

* Priority

Action
 Allow Deny

Description

* Subscription

* Virtual Network

* Subnet

To delete a rule, click the ... on your rule and then click **remove**.

Blocking a single IP Address

When adding your first IP Restriction rule, the service will add an explicit **deny all** rule with a priority of 2147483647. In practice, the explicit **deny all** rule will be last rule executed and will block access to any IP address that is not explicitly allowed using an **Allow** rule.

For the scenario where users want to explicitly block a single IP address or IP address block, but allow everything else access, it is necessary to add an explicit **Allow All** rule.

SCM site

In addition to being able to control access to your app, you can also restrict access to the scm site used by your app. The scm site is the web deploy endpoint and also the Kudu console. You can separately assign access restrictions to the scm site from the app or use the same set for both the app and the scm site. When you check the box to have the same restrictions as your app, everything is blanked out. If you uncheck the box, whatever settings you had earlier on the scm site are applied.

PRIORITY	NAME	SOURCE	ENDPOINT STATUS	ACTION
1	Allow all	Any	Allow	Allow

Programmatic manipulation of access restriction rules

Azure CLI and Azure PowerShell has support for editing access restrictions. Example of adding an access restriction using Azure CLI:

```
az webapp config access-restriction add --resource-group ResourceGroup --name AppName \
--rule-name 'IP example rule' --action Allow --ip-address 122.133.144.0/24 --priority 100
```

Example of adding an access restriction using Azure PowerShell:

```
Add-AzWebAppAccessRestrictionRule -ResourceGroupName "ResourceGroup" -WebAppName "AppName"
-Name "Ip example rule" -Priority 100 -Action Allow -IpAddress 122.133.144.0/24
```

Values can also be set manually with an [Azure REST API](#) PUT operation on the app configuration in Resource Manager or using an Azure Resource Manager template. As an example, you can use [resources.azure.com](#) and edit the ipSecurityRestrictions block to add the required JSON.

The location for this information in Resource Manager is:

management.azure.com/subscriptions/{subscription ID}/resourceGroups/{resource groups}/providers/Microsoft.Web/sites/{web app name}/config/web?api-version=2018-02-01

The JSON syntax for the earlier example is:

```
{
  "properties": {
    "ipSecurityRestrictions": [
      {
        "ipAddress": "122.133.144.0/24",
        "action": "Allow",
        "priority": 100,
        "name": "IP example rule"
      }
    ]
  }
}
```

Azure Function App Access Restrictions

Access restrictions are available for both Function Apps with the same functionality as App Service plans. Enabling access restrictions will disable the portal code editor for any disallowed IPs.

Next steps

[Access restrictions for Azure Function Apps](#)

[Application Gateway integration with service endpoints](#)

How to use managed identities for App Service and Azure Functions

2/12/2020 • 12 minutes to read • [Edit Online](#)

IMPORTANT

Managed identities for App Service and Azure Functions will not behave as expected if your app is migrated across subscriptions/tenants. The app will need to obtain a new identity, which can be done by disabling and re-enabling the feature. See [Removing an identity](#) below. Downstream resources will also need to have access policies updated to use the new identity.

This topic shows you how to create a managed identity for App Service and Azure Functions applications and how to use it to access other resources. A managed identity from Azure Active Directory (AAD) allows your app to easily access other AAD-protected resources such as Azure Key Vault. The identity is managed by the Azure platform and does not require you to provision or rotate any secrets. For more about managed identities in AAD, see [Managed identities for Azure resources](#).

Your application can be granted two types of identities:

- A **system-assigned identity** is tied to your application and is deleted if your app is deleted. An app can only have one system-assigned identity.
- A **user-assigned identity** is a standalone Azure resource which can be assigned to your app. An app can have multiple user-assigned identities.

Adding a system-assigned identity

Creating an app with a system-assigned identity requires an additional property to be set on the application.

Using the Azure portal

To set up a managed identity in the portal, you will first create an application as normal and then enable the feature.

1. Create an app in the portal as you normally would. Navigate to it in the portal.
2. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.
3. Select **Identity**.
4. Within the **System assigned** tab, switch **Status** to **On**. Click **Save**.

The screenshot shows the Azure portal interface for managing an App Service. The URL in the address bar is <https://portal.azure.com/#@contoso.onmicrosoft.com/resource/subscriptions/01234567-89ab-cdef-0123-456789abcdef/resourceGroups...>. The user meganb@contoso.com is logged in. The main navigation bar shows 'Microsoft Azure' and the current page is 'App Services > systemassigned-linux - Identity'. On the left, there's a sidebar with links like 'Authentication / Authorization', 'Application Insights', 'Identity' (which is highlighted with a red box), 'Backups', 'Custom domains', 'TLS/SSL settings', 'Networking', and 'Scale up (App Service plan)'. The main content area has tabs for 'System assigned' (which is selected and highlighted with a red box) and 'User assigned'. Below the tabs is a description of what a system assigned managed identity is. At the bottom of the main content area are buttons for 'Save', 'Discard', 'Refresh', and 'Got feedback?'. Under the 'Status' heading, there's a toggle switch that is currently set to 'On' (highlighted with a red box). The status text says 'A system assigned managed identity enables Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based-access-control. The lifecycle of this type of managed identity is tied to the lifecycle of this resource. Additionally, each resource (e.g. Virtual Machine) can only have one system assigned managed identity.' There's also a 'Learn more about Managed identities' link.

Using the Azure CLI

To set up a managed identity using the Azure CLI, you will need to use the `az webapp identity assign` command against an existing application. You have three options for running the examples in this section:

- Use [Azure Cloud Shell](#) from the Azure portal.
- Use the embedded Azure Cloud Shell via the "Try It" button, located in the top right corner of each code block below.
- [Install the latest version of Azure CLI](#) (2.0.31 or later) if you prefer to use a local CLI console.

The following steps will walk you through creating a web app and assigning it an identity using the CLI:

1. If you're using the Azure CLI in a local console, first sign in to Azure using `az login`. Use an account that is associated with the Azure subscription under which you would like to deploy the application:

```
az login
```

2. Create a web application using the CLI. For more examples of how to use the CLI with App Service, see [App Service CLI samples](#):

```
az group create --name myResourceGroup --location westus
az appservice plan create --name myPlan --resource-group myResourceGroup --sku S1
az webapp create --name myApp --resource-group myResourceGroup --plan myPlan
```

3. Run the `identity assign` command to create the identity for this application:

```
az webapp identity assign --name myApp --resource-group myResourceGroup
```

Using Azure PowerShell

NOTE

This article has been updated to use the 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 steps will walk you through creating a web app and assigning it an identity using Azure PowerShell:

1. If needed, install the Azure PowerShell using the instructions found in the [Azure PowerShell guide](#), and then run `Login-AzAccount` to create a connection with Azure.
2. Create a web application using Azure PowerShell. For more examples of how to use Azure PowerShell with App Service, see [App Service PowerShell samples](#):

```
# Create a resource group.  
New-AzResourceGroup -Name myResourceGroup -Location $location  
  
# Create an App Service plan in Free tier.  
New-AzAppServicePlan -Name $webappname -Location $location -ResourceGroupName myResourceGroup -Tier Free  
  
# Create a web app.  
New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname -ResourceGroupName myResourceGroup
```

3. Run the `Set-AzWebApp -AssignIdentity` command to create the identity for this application:

```
Set-AzWebApp -AssignIdentity $true -Name $webappname -ResourceGroupName myResourceGroup
```

Using an Azure Resource Manager template

An Azure Resource Manager template can be used to automate deployment of your Azure resources. To learn more about deploying to App Service and Functions, see [Automating resource deployment in App Service](#) and [Automating resource deployment in Azure Functions](#).

Any resource of type `Microsoft.Web/sites` can be created with an identity by including the following property in the resource definition:

```
"identity": {  
    "type": "SystemAssigned"  
}
```

NOTE

An application can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned,UserAssigned`

Adding the system-assigned type tells Azure to create and manage the identity for your application.

For example, a web app might look like the following:

```
{
    "apiVersion": "2016-08-01",
    "type": "Microsoft.Web/sites",
    "name": "[variables('appName')]",
    "location": "[resourceGroup().location]",
    "identity": {
        "type": "SystemAssigned"
    },
    "properties": {
        "name": "[variables('appName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]",
        "hostingEnvironment": "",
        "clientAffinityEnabled": false,
        "alwaysOn": true
    },
    "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]"
    ]
}
```

When the site is created, it has the following additional properties:

```
"identity": {
    "type": "SystemAssigned",
    "tenantId": "<TENANTID>",
    "principalId": "<PRINCIPALID>"
}
```

Where `<TENANTID>` and `<PRINCIPALID>` are replaced with GUIDs. The tenantId property identifies what AAD tenant the identity belongs to. The principalId is a unique identifier for the application's new identity. Within AAD, the service principal has the same name that you gave to your App Service or Azure Functions instance.

Adding a user-assigned identity

Creating an app with a user-assigned identity requires that you create the identity and then add its resource identifier to your app config.

Using the Azure portal

First, you'll need to create a user-assigned identity resource.

1. Create a user-assigned managed identity resource according to [these instructions](#).
2. Create an app in the portal as you normally would. Navigate to it in the portal.
3. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.
4. Select **Identity**.
5. Within the **User assigned** tab, click **Add**.
6. Search for the identity you created earlier and select it. Click **Add**.

The screenshot shows the Azure portal interface for managing identities. On the left, there's a sidebar with options like 'Search (Ctrl+)', 'Settings', 'Configuration', 'Authentication / Authorizati...', 'Application Insights', 'Identity' (which is highlighted with a red box), 'Backups', 'Custom domains', and 'TLS/SSL settings'. The main area has tabs for 'System assigned' and 'User assigned', with 'User assigned' being the active one. Below the tabs, there's a brief description of user-assigned managed identities. A search bar at the top right contains the text 'userassigned'. A list of identities is shown, with one entry, 'userassignedmanagedidentity', selected and highlighted with a red box. At the bottom right of the list area, there's a blue 'Add' button, which is also highlighted with a red box.

Using an Azure Resource Manager template

An Azure Resource Manager template can be used to automate deployment of your Azure resources. To learn more about deploying to App Service and Functions, see [Automating resource deployment in App Service](#) and [Automating resource deployment in Azure Functions](#).

Any resource of type `Microsoft.Web/sites` can be created with an identity by including the following block in the resource definition, replacing `<RESOURCEID>` with the resource ID of the desired identity:

```
"identity": {  
    "type": "UserAssigned",  
    "userAssignedIdentities": {  
        "<RESOURCEID>": {}  
    }  
}
```

NOTE

An application can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned,UserAssigned`

Adding the user-assigned type tells Azure to use the user-assigned identity specified for your application.

For example, a web app might look like the following:

```
{
    "apiVersion": "2016-08-01",
    "type": "Microsoft.Web/sites",
    "name": "[variables('appName')]",
    "location": "[resourceGroup().location]",
    "identity": {
        "type": "UserAssigned",
        "userAssignedIdentities": {
            "[resourceId('Microsoft.ManagedIdentity/userAssignedIdentities', variables('identityName'))]": {}
        }
    },
    "properties": {
        "name": "[variables('appName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]",
        "hostingEnvironment": "",
        "clientAffinityEnabled": false,
        "alwaysOn": true
    },
    "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]",
        "[resourceId('Microsoft.ManagedIdentity/userAssignedIdentities', variables('identityName'))]"
    ]
}
```

When the site is created, it has the following additional properties:

```
"identity": {
    "type": "UserAssigned",
    "userAssignedIdentities": {
        "<RESOURCEID>": {
            "principalId": "<PRINCIPALID>",
            "clientId": "<CLIENTID>"
        }
    }
}
```

Where `<PRINCIPALID>` and `<CLIENTID>` are replaced with GUIDs. The principalId is a unique identifier for the identity which is used for AAD administration. The clientId is a unique identifier for the application's new identity that is used for specifying which identity to use during runtime calls.

Obtaining tokens for Azure resources

An app can use its managed identity to get tokens to access other resources protected by AAD, such as Azure Key Vault. These tokens represent the application accessing the resource, and not any specific user of the application.

IMPORTANT

You may need to configure the target resource to allow access from your application. For example, if you request a token to access Key Vault, you need to make sure you have added an access policy that includes your application's identity. Otherwise, your calls to Key Vault will be rejected, even if they include the token. To learn more about which resources support Azure Active Directory tokens, see [Azure services that support Azure AD authentication](#).

There is a simple REST protocol for obtaining a token in App Service and Azure Functions. This can be used for all applications and languages. For .NET and Java, the Azure SDK provides an abstraction over this protocol and facilitates a local development experience.

Using the REST protocol

An app with a managed identity has two environment variables defined:

- MSI_ENDPOINT - the URL to the local token service.
- MSI_SECRET - a header used to help mitigate server-side request forgery (SSRF) attacks. The value is rotated by the platform.

The **MSI_ENDPOINT** is a local URL from which your app can request tokens. To get a token for a resource, make an HTTP GET request to this endpoint, including the following parameters:

PARAMETER NAME	IN	DESCRIPTION
resource	Query	The AAD resource URI of the resource for which a token should be obtained. This could be one of the Azure services that support Azure AD authentication or any other resource URI.
api-version	Query	The version of the token API to be used. "2017-09-01" is currently the only version supported.
secret	Header	The value of the MSI_SECRET environment variable. This header is used to help mitigate server-side request forgery (SSRF) attacks.
clientid	Query	(Optional unless for user-assigned) The ID of the user-assigned identity to be used. If omitted, the system-assigned identity is used.

IMPORTANT

If you are attempting to obtain tokens for user-assigned identities, you must include the `clientid` property. Otherwise the token service will attempt to obtain a token for a system-assigned identity, which may or may not exist.

A successful 200 OK response includes a JSON body with the following properties:

PROPERTY NAME	DESCRIPTION
access_token	The requested access token. The calling web service can use this token to authenticate to the receiving web service.
expires_on	The time when the access token expires. The date is represented as the number of seconds from 1970-01-01T0:0:0Z UTC until the expiration time. This value is used to determine the lifetime of cached tokens.
resource	The App ID URI of the receiving web service.
token_type	Indicates the token type value. The only type that Azure AD supports is Bearer. For more information about bearer tokens, see The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC 6750) .

This response is the same as the [response for the AAD service-to-service access token request](#).

NOTE

Environment variables are set up when the process first starts, so after enabling a managed identity for your application, you may need to restart your application, or redeploy its code, before `MSI_ENDPOINT` and `MSI_SECRET` are available to your code.

REST protocol examples

An example request might look like the following:

```
GET /MSI/token?resource=https://vault.azure.net&api-version=2017-09-01 HTTP/1.1
Host: localhost:4141
Secret: 853b9a84-5bfa-4b22-a3f3-0b9a43d9ad8a
```

And a sample response might look like the following:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "eyJ0eXAi...",
  "expires_on": "09/14/2017 00:00:00 PM +00:00",
  "resource": "https://vault.azure.net",
  "token_type": "Bearer"
}
```

Code examples

- [.NET](#)
- [JavaScript](#)
- [Python](#)
- [PowerShell](#)

TIP

For .NET languages, you can also use [Microsoft.Azure.Services.AppAuthentication](#) instead of crafting this request yourself.

```
private readonly HttpClient _client;
// ...
public async Task<HttpResponseMessage> GetToken(string resource) {
    var request = new HttpRequestMessage(HttpMethod.Get,
        String.Format("{0}/?resource={1}&api-version=2017-09-01",
        Environment.GetEnvironmentVariable("MSI_ENDPOINT"), resource));
    request.Headers.Add("Secret", Environment.GetEnvironmentVariable("MSI_SECRET"));
    return await _client.SendAsync(request);
}
```

Using the Microsoft.Azure.Services.AppAuthentication library for .NET

For .NET applications and functions, the simplest way to work with a managed identity is through the [Microsoft.Azure.Services.AppAuthentication](#) package. This library will also allow you to test your code locally on your development machine, using your user account from Visual Studio, the [Azure CLI](#), or Active Directory Integrated Authentication. For more on local development options with this library, see the [Microsoft.Azure.Services.AppAuthentication reference](#). This section shows you how to get started with the library

in your code.

1. Add references to the [Microsoft.Azure.Services.AppAuthentication](#) and any other necessary NuGet packages to your application. The below example also uses [Microsoft.Azure.KeyVault](#).
2. Add the following code to your application, modifying to target the correct resource. This example shows two ways to work with Azure Key Vault:

```
using Microsoft.Azure.Services.AppAuthentication;
using Microsoft.Azure.KeyVault;
// ...
var azureServiceTokenProvider = new AzureServiceTokenProvider();
string accessToken = await azureServiceTokenProvider.GetAccessTokenAsync("https://vault.azure.net");
// OR
var kv = new KeyVaultClient(new
    KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaultTokenCallback));
```

To learn more about Microsoft.Azure.Services.AppAuthentication and the operations it exposes, see the [Microsoft.Azure.Services.AppAuthentication reference](#) and the [App Service and KeyVault with MSI .NET sample](#).

Using the Azure SDK for Java

For Java applications and functions, the simplest way to work with a managed identity is through the [Azure SDK for Java](#). This section shows you how to get started with the library in your code.

1. Add a reference to the [Azure SDK library](#). For Maven projects, you might add this snippet to the `dependencies` section of the project's POM file:

```
<dependency>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>azure</artifactId>
    <version>1.23.0</version>
</dependency>
```

2. Use the `AppServiceMSICredentials` object for authentication. This example shows how this mechanism may be used for working with Azure Key Vault:

```
import com.microsoft.azure.AzureEnvironment;
import com.microsoft.azure.management.Azure;
import com.microsoft.azure.management.keyvault.Vault
//...
Azure azure = Azure.authenticate(new AppServiceMSICredentials(AzureEnvironment.AZURE))
    .withSubscription(subscriptionId);
Vault myKeyVault = azure.vaults().getByResourceGroup(resourceGroup, keyvaultName);
```

Removing an identity

A system-assigned identity can be removed by disabling the feature using the portal, PowerShell, or CLI in the same way that it was created. User-assigned identities can be removed individually. To remove all identities, in the REST/ARM template protocol, this is done by setting the type to "None":

```
"identity": {
    "type": "None"
}
```

Removing a system-assigned identity in this way will also delete it from AAD. System-assigned identities are also

automatically removed from AAD when the app resource is deleted.

NOTE

There is also an application setting that can be set, WEBSITE_DISABLE_MSI, which just disables the local token service. However, it leaves the identity in place, and tooling will still show the managed identity as "on" or "enabled." As a result, use of this setting is not recommended.

Next steps

[Access SQL Database securely using a managed identity](#)

Use Key Vault references for App Service and Azure Functions

1/8/2020 • 4 minutes to read • [Edit Online](#)

This topic shows you how to work with secrets from Azure Key Vault in your App Service or Azure Functions application without requiring any code changes. [Azure Key Vault](#) is a service that provides centralized secrets management, with full control over access policies and audit history.

Granting your app access to Key Vault

In order to read secrets from Key Vault, you need to have a vault created and give your app permission to access it.

1. Create a key vault by following the [Key Vault quickstart](#).
2. Create a [system-assigned managed identity](#) for your application.

NOTE

Key Vault references currently only support system-assigned managed identities. User-assigned identities cannot be used.

3. Create an [access policy in Key Vault](#) for the application identity you created earlier. Enable the "Get" secret permission on this policy. Do not configure the "authorized application" or `applicationId` settings, as this is not compatible with a managed identity.

NOTE

Key Vault references are not presently able to resolve secrets stored in a key vault with [network restrictions](#).

Reference syntax

A Key Vault reference is of the form `@Microsoft.KeyVault({referenceString})`, where `{referenceString}` is replaced by one of the following options:

REFERENCE STRING	DESCRIPTION
<code>SecretUri=secretUri</code>	The SecretUri should be the full data-plane URI of a secret in Key Vault, including a version, e.g., https://myvault.vault.azure.net/secrets/mysecret/ec96f02080254f109c51a1f14cdb1931
<code>VaultName=vaultName;SecretName=secretName;SecretVersion=secretVersion</code>	The VaultName should be the name of your Key Vault resource. The SecretName should be the name of the target secret. The SecretVersion should be the version of the secret to use.

For example, a complete reference with Version would look like the following:

```
@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/ec96f02080254f109c51a1f14cdb1931)
```

Alternatively:

```
@Microsoft.KeyVault(VaultName=myvault;SecretName=mysecret;SecretVersion=ec96f02080254f109c51a1f14cdb1931)
```

Source Application Settings from Key Vault

Key Vault references can be used as values for [Application Settings](#), allowing you to keep secrets in Key Vault instead of the site config. Application Settings are securely encrypted at rest, but if you need secret management capabilities, they should go into Key Vault.

To use a Key Vault reference for an application setting, set the reference as the value of the setting. Your app can reference the secret through its key as normal. No code changes are required.

TIP

Most application settings using Key Vault references should be marked as slot settings, as you should have separate vaults for each environment.

Azure Resource Manager deployment

When automating resource deployments through Azure Resource Manager templates, you may need to sequence your dependencies in a particular order to make this feature work. Of note, you will need to define your application settings as their own resource, rather than using a `siteConfig` property in the site definition. This is because the site needs to be defined first so that the system-assigned identity is created with it and can be used in the access policy.

An example psuedo-template for a function app might look like the following:

```
{
  //...
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageAccountName')]",
      //...
    },
    {
      "type": "Microsoft.Insights/components",
      "name": "[variables('appInsightsName')]",
      //...
    },
    {
      "type": "Microsoft.Web/sites",
      "name": "[variables('functionAppName')]",
      "identity": {
        "type": "SystemAssigned"
      },
      //...
    },
    "resources": [
      {
        "type": "config",
        "name": "appsettings",
        //...
        "dependsOn": [
          "[resourceId('Microsoft.Web/sites', variables('functionAppName'))]",
          "[resourceId('Microsoft.KeyVault/vaults/', variables('keyVaultName'))]",
          "[resourceId('Microsoft.KeyVault/vaults/secrets', variables('keyVaultName')),
variables('storageConnectionStringName'))]",
          "[resourceId('Microsoft.KeyVault/vaults/secrets', variables('keyVaultName'),
variables('appInsightsKeyName'))]"
        ],
        1.
    
```

```

    "properties": {
        "AzureWebJobsStorage": "[concat('@Microsoft.KeyVault(SecretUri=',
reference(variables('storageConnectionStringResourceId')).secretUriWithVersion, ')'))]",
        "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING": "[concat('@Microsoft.KeyVault(SecretUri=',
reference(variables('storageConnectionStringResourceId')).secretUriWithVersion, ')'))]",
        "APPINSIGHTS_INSTRUMENTATIONKEY": "[concat('@Microsoft.KeyVault(SecretUri=',
reference(variables('appInsightsKeyResourceId')).secretUriWithVersion, ')'))]",
        "WEBSITE_ENABLE_SYNC_UPDATE_SITE": "true"
        //...
    }
},
{
    "type": "sourcecontrols",
    "name": "web",
    //...
    "dependsOn": [
        "[resourceId('Microsoft.Web/sites', variables('functionAppName'))]",
        "[resourceId('Microsoft.Web/sites/config', variables('functionAppName')),
'appsettings')]"
    ],
    }
],
},
{
    "type": "Microsoft.KeyVault/vaults",
    "name": "[variables('keyVaultName')]",
    //...
    "dependsOn": [
        "[resourceId('Microsoft.Web/sites', variables('functionAppName'))]"
    ],
    "properties": {
        //...
        "accessPolicies": [
            {
                "tenantId": "[reference(concat('Microsoft.Web/sites/', variables('functionAppName'),
'/providers/Microsoft.ManagedIdentity/Identities/default'), '2015-08-31-PREVIEW').tenantId]",
                "objectId": "[reference(concat('Microsoft.Web/sites/', variables('functionAppName'),
'/providers/Microsoft.ManagedIdentity/Identities/default'), '2015-08-31-PREVIEW').principalId]",
                "permissions": {
                    "secrets": [ "get" ]
                }
            }
        ]
    },
    "resources": [
        {
            "type": "secrets",
            "name": "[variables('storageConnectionStringName')]",
            //...
            "dependsOn": [
                "[resourceId('Microsoft.KeyVault/vaults/', variables('keyVaultName'))]",
                "[resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName'))]"
            ],
            "properties": {
                "value": "[concat('DefaultEndpointsProtocol=https;AccountName=', variables('storageAccountName'), ';AccountKey=', listKeys(variables('storageAccountResourceId'), '2015-05-01-preview').key1)]"
            }
        },
        {
            "type": "secrets",
            "name": "[variables('appInsightsKeyName')]",
            //...
            "dependsOn": [
                "[resourceId('Microsoft.KeyVault/vaults/', variables('keyVaultName'))]",
                "[resourceId('Microsoft.Insights/components', variables('appInsightsName'))]"
            ],
            "properties": {
                "value": "[reference(resourceId('microsoft.insights/components/',

```

```
        value : [reference('secretName', 'Microsoft.KeyVaultComponents'),  
        variables('appInsightsName')), '2015-05-01').InstrumentationKey]  
    }  
}  
}  
}  
}
```

NOTE

In this example, the source control deployment depends on the application settings. This is normally unsafe behavior, as the app setting update behaves asynchronously. However, because we have included the `WEBSITE_ENABLE_SYNC_UPDATE_SITE` application setting, the update is synchronous. This means that the source control deployment will only begin once the application settings have been fully updated.

Troubleshooting Key Vault References

If a reference is not resolved properly, the reference value will be used instead. This means that for application settings, an environment variable would be created whose value has the `@Microsoft.KeyVault(...)` syntax. This may cause the application to throw errors, as it was expecting a secret of a certain structure.

Most commonly, this is due to a misconfiguration of the [Key Vault access policy](#). However, it could also be due to a secret no longer existing or a syntax error in the reference itself.

If the syntax is correct, you can view other causes for error by checking the current resolution status in the portal. Navigate to Application Settings and select "Edit" for the reference in question. Below the setting configuration, you should see status information, including any errors. The absence of these implies that the reference syntax is invalid.

You can also use one of the built-in detectors to get additional information.

Using the detector for App Service

1. In the portal, navigate to your app.
2. Select **Diagnose and solve problems**.
3. Choose **Availability and Performance** and select **Web app down**.
4. Find **Key Vault Application Settings Diagnostics** and click **More info**.

Using the detector for Azure Functions

1. In the portal, navigate to your app.
2. Navigate to **Platform features**.
3. Select **Diagnose and solve problems**.
4. Choose **Availability and Performance** and select **Function app down or reporting errors**.
5. Click on **Key Vault Application Settings Diagnostics**.

Use an SSL certificate in your code in Azure App Service

12/2/2019 • 3 minutes to read • [Edit Online](#)

In your application code, you can access the [public or private certificates you add to App Service](#). Your app code may act as a client and access an external service that requires certificate authentication, or it may need to perform cryptographic tasks. This how-to guide shows how to use public or private certificates in your application code.

This approach to using certificates in your code makes use of the SSL functionality in App Service, which requires your app to be in **Basic** tier or above. If your app is in **Free** or **Shared** tier, you can [include the certificate file in your app repository](#).

When you let App Service manage your SSL certificates, you can maintain the certificates and your application code separately and safeguard your sensitive data.

Prerequisites

To follow this how-to guide:

- [Create an App Service app](#)
- [Add a certificate to your app](#)

Find the thumbprint

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, select **TLS/SSL settings**, then select **Private Key Certificates (.pfx)** or **Public Key Certificates (.cer)**.

Find the certificate you want to use and copy the thumbprint.

Private Key Certificates				
Status Filter				
All	Healthy	Warning	Expired	
Health Status	Hostname		Expiration	Thumbprint
 Healthy	www.contoso.com		4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...
				...

Make the certificate accessible

To access a certificate in your app code, add its thumbprint to the `WEBSITE_LOAD_CERTIFICATES` app setting, by running the following command in the [Cloud Shell](#):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings WEBSITE_LOAD_CERTIFICATES=<comma-separated-certificate-thumbprints>
```

To make all your certificates accessible, set the value to `*`.

Load certificate in Windows apps

The `WEBSITE_LOAD_CERTIFICATES` app setting makes the specified certificates accessible to your Windows hosted app in the Windows certificate store, and the location depends on the [pricing tier](#):

- **Isolated** tier - in [Local Machine\My](#).
- All other tiers - in [Current User\My](#).

In C# code, you access the certificate by the certificate thumbprint. The following code loads a certificate with the thumbprint `E661583E8FABEF4C0BEF694CBC41C28FB81CD870`.

```
using System;
using System.Security.Cryptography.X509Certificates;

...
X509Store certStore = new X509Store(StoreName.My, StoreLocation.CurrentUser);
certStore.Open(OpenFlags.ReadOnly);
X509Certificate2Collection certCollection = certStore.Certificates.Find(
    X509FindType.FindByThumbprint,
    // Replace below with your certificate's thumbprint
    "E661583E8FABEF4C0BEF694CBC41C28FB81CD870",
    false);
// Get the first cert with the thumbprint
if (certCollection.Count > 0)
{
    X509Certificate2 cert = certCollection[0];
    // Use certificate
    Console.WriteLine(cert.FriendlyName);
}
certStore.Close();
...
```

In Java code, you access the certificate from the "Windows-MY" store using the Subject Common Name field (see [Public key certificate](#)). The following code shows how to load a private key certificate:

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import java.security.KeyStore;
import java.security.cert.Certificate;
import java.security.PrivateKey;

...
KeyStore ks = KeyStore.getInstance("Windows-MY");
ks.load(null, null);
Certificate cert = ks.getCertificate("<subject-cn>");
PrivateKey privKey = (PrivateKey) ks.getKey("<subject-cn>", ("<password>").toCharArray());

// Use the certificate and key
...
```

For languages that don't support or offer insufficient support for the Windows certificate store, see [Load certificate from file](#).

Load certificate in Linux apps

The `WEBSITE_LOAD_CERTIFICATES` app settings makes the specified certificates accessible to your Linux hosted apps (including custom container apps) as files. The files are found under the following directories:

- Private certificates - `/var/ssl/private` (`.p12` files)
- Public certificates - `/var/ssl/certs` (`.der` files)

The certificate file names are the certificate thumbprints. The following C# code shows how to load a public certificate in a Linux app.

```
using System;
using System.Security.Cryptography.X509Certificates;

...
var bytes = System.IO.File.ReadAllBytes("/var/ssl/certs/<thumbprint>.der");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate
```

To see how to load an SSL certificate from a file in Node.js, PHP, Python, Java, or Ruby, see the documentation for the respective language or web platform.

Load certificate from file

If you need to load a certificate file that you upload manually, it's better to upload the certificate using [FTPS](#) instead of [Git](#), for example. You should keep sensitive data like a private certificate out of source control.

NOTE

ASP.NET and ASP.NET Core on Windows must access the certificate store even if you load a certificate from a file. To load a certificate file in a Windows .NET app, load the current user profile with the following command in the [Cloud Shell](#):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
WEBSITE_LOAD_USER_PROFILE=1
```

This approach to using certificates in your code makes use of the SSL functionality in App Service, which requires your app to be in **Basic** tier or above.

The following C# example loads a public certificate from a relative path in your app:

```
using System;
using System.Security.Cryptography.X509Certificates;

...
var bytes = System.IO.File.ReadAllBytes("~/<relative-path-to-cert-file>");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate
```

To see how to load an SSL certificate from a file in Node.js, PHP, Python, Java, or Ruby, see the documentation for the respective language or web platform.

More resources

- [Secure a custom DNS name with an SSL binding](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [FAQ : App Service Certificates](#)

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

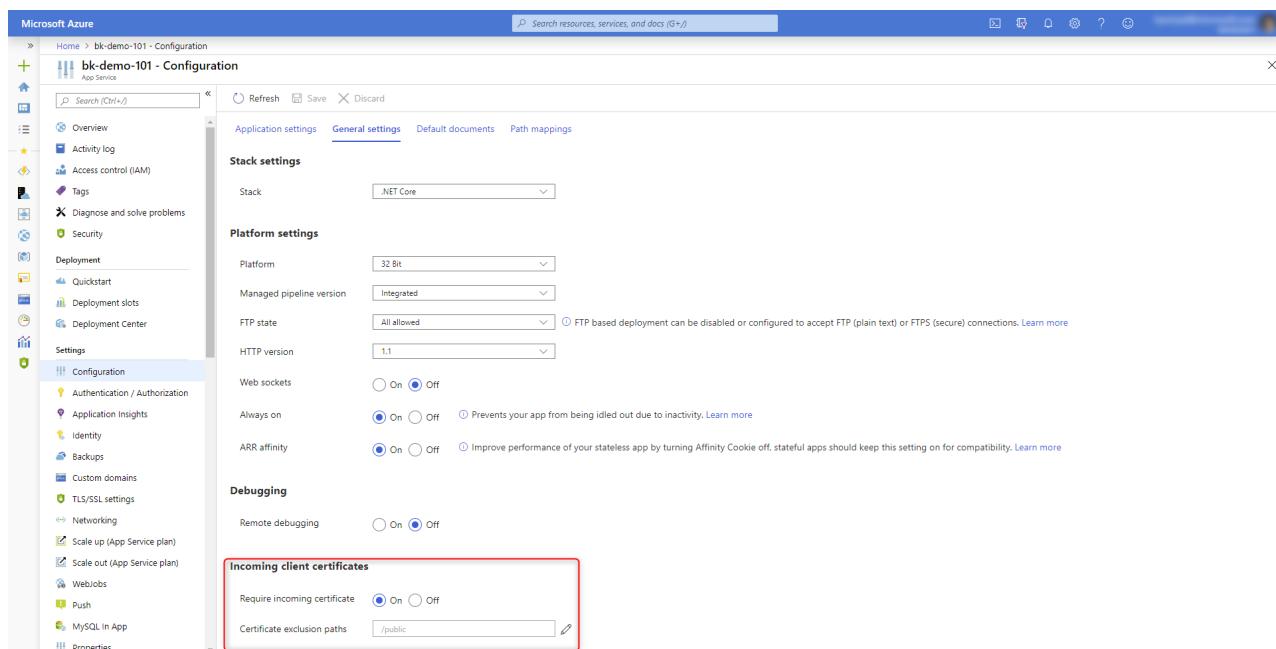
To set up your app to require client certificates, you need to set the `clientCertEnabled` setting for your app to `true`. To set the setting, run the following command in the [Cloud Shell](#).

```
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 'Scale up (App Service plan)' configuration page. The left sidebar includes options like Identity, Backups, Custom domains, TLS/SSL settings, Networking, Scale up (App Service plan) (highlighted), Scale out (App Service plan), WebJobs, Push, MySQL In App, Properties, Locks, Export template, App Service plan, Quotas, and Change App Service plan. The main area displays three recommended pricing tiers: F1 (Free), D1 (Shared infrastructure, 1 GB memory, 60 minutes/day compute), and B1 (100 total ACU, 1.75 GB memory, A-Series compute equivalent). The 'Included features' section lists Custom domains, and the 'Included hardware' section details Azure Compute Units (ACU), Memory (1 GB), and Storage (1 GB disk storage). An 'Apply' button is located 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 links for Overview (highlighted), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, and Deployment. The main area displays a summary with a 'Resource group /change' link (highlighted), myResourceGroup, Status (Running), Location (West Europe), Subscription (change) mySubscription, Subscription ID 00000000-0000-0000-000000000000, URL https://my-demo-a..., App Service Plan myAppServicePlan, FTP/deployment user my-demo-app\user, FTP hostname ftp://waws-prod-an..., and FTPS hostname https://waws-prod-an...

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](#)

Configure PremiumV2 tier for Azure App Service

12/2/2019 • 4 minutes to read • [Edit Online](#)

The new **PremiumV2** pricing tier gives you faster processors, SSD storage, and doubles the memory-to-core ratio of the existing pricing tiers. With the performance advantage, you could save money by running your apps on fewer instances. In this article, you learn how to create an app in **PremiumV2** tier or scale up an app to **PremiumV2** tier.

Prerequisites

To scale-up an app to **PremiumV2**, you need to have an Azure App Service app that runs in a pricing tier lower than **PremiumV2**, and the app must be running in an App Service deployment that supports PremiumV2.

PremiumV2 availability

The **PremiumV2** tier is available for App Service on both *Windows* as well as *Linux*.

PremiumV2 is available in most Azure regions. To see if it's available in your region, run the following Azure CLI command in the [Azure Cloud Shell](#):

```
az appservice list-locations --sku P1V2
```

Create an app in PremiumV2 tier

The pricing tier of an App Service app is defined in the [App Service plan](#) that it runs on. You can create an App Service plan by itself or as part of app creation.

When configuring the App Service plan in the [Azure portal](#), select **Pricing tier**.

Select **Production**, then select **P1V2**, **P2V2**, or **P3V2**, then click **Apply**.


Dev / Test
For less demanding workloads


Production
For most production workloads


Isolated
Advanced networking and scale

Recommended pricing tiers

S1 1x cores 1.75 GB memory A-Series compute 74.40 USD/Month (Estimated)	P1v2 1x cores 3.5 GB memory Dv2-Series compute 148.80 USD/Month (Estimated)	P2v2 2x cores 7 GB memory Dv2-Series compute 297.60 USD/Month (Estimated)
▼ See additional options		

Included features

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

-  **Custom domains / SSL**
Configure and purchase custom domains with SNI and IP SSL bindings
-  **Auto scale**
Up to 20 instances. Subject to availability.
-  **Staging slots**
Up to 20 staging slots to use for testing and deployments before swapping the...
-  **Daily backups**
Backup your app 50 times daily.
-  **Traffic manager**
Improve performance and availability by routing traffic between multiple instanc...

Included hardware

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

-  **CPU**
Dedicated Dv2-series compute resources used to run applications deployed in th...
-  **Memory**
Memory per instance available to run applications deployed and running in th...
-  **Storage**
250 GB disk storage shared by all apps deployed in the App Service plan.

Apply

IMPORTANT

If you don't see **P1V2**, **P2V2**, and **P3V2** as options, or if the options are greyed out, then **PremiumV2** likely isn't available in the underlying App Service deployment that contains the App Service plan. See [Scale up from an unsupported resource group and region combination](#) for more details.

Scale up an existing app to PremiumV2 tier

Before scaling an existing app to **PremiumV2** tier, make sure that **PremiumV2** is available. For information, see [PremiumV2 availability](#). If it's not available, see [Scale up from an unsupported resource group and region combination](#).

Depending on your hosting environment, scaling up may require extra steps.

In the [Azure portal](#), open your App Service app page.

In the left navigation of your App Service app page, select **Scale up (App Service plan)**.

The screenshot shows the Azure App Service overview page for the app "app-scaling-premiumv2". On the left, there's a sidebar with various settings like Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs, and Push. The "Scale up (App Service plan)" option is highlighted with a red box. The main pane displays basic information: Resource group (myResourceGroupWin), Status (Running), Location (West Europe), Subscription (change), and Subscription ID (redacted). Below this is a preview section showing "Http 5xx" and "Data" with a value of 100.

Select **Production**, then select **P1V2**, **P2V2**, or **P3V2**, then click **Apply**.

This screenshot shows the "Scale up (App Service plan)" configuration page. It starts with three tabs: "Dev / Test" (for less demanding workloads), "Production" (selected and highlighted with a red box, described as "For most production workloads"), and "Isolated" (advanced networking and scale). Below the tabs, it lists recommended pricing tiers:

Tier	Cores	Memory	Compute Series	Price (Estimated)
S1	1x cores	1.75 GB memory	A-Series compute	74.40 USD/Month (Estimated)
P1v2	1x cores	3.5 GB memory	Dv2-Series compute	148.80 USD/Month (Estimated)
P2v2	2x cores	7 GB memory	Dv2-Series compute	297.60 USD/Month (Estimated)
P3v2	4x cores	14 GB memory	Dv2-Series compute	595.20 USD/Month (Estimated)

Below the tiers, there's a "See additional options" link. The page then details included features and hardware:

Included features:
Custom domains / SSL, Auto scale, Staging slots, Daily backups, Traffic manager.

Included hardware:
CPU, Memory, Storage.

At the bottom, there's an "Apply" button highlighted with a red box.

If your operation finishes successfully, your app's overview page shows that it's now in a **PremiumV2** tier.

Click here to access our Quickstart guide for deploying code to your app →

Resource group (change) myResourceGroupWin	URL http://app-scaling-premiumv2.azurewebsites.net
Status Running	App Service plan/pricing tier app-scaling-premiumv2 (PremiumV2: 1 Small)
Location West Europe	FTP/deployment username [REDACTED]
Subscription (change) [REDACTED]	FTP hostname [REDACTED]
Subscription ID [REDACTED]	FTPS hostname [REDACTED]

If you get an error

Some App Service plans can't scale up to the PremiumV2 tier if the underlying App Service deployment doesn't support PremiumV2. See [Scale up from an unsupported resource group and region combination](#) for more details.

Scale up from an unsupported resource group and region combination

If your app runs in an App Service deployment where **PremiumV2** isn't available, or if your app runs in a region that currently does not support **PremiumV2**, you will need to re-deploy your app to take advantage of

PremiumV2. You have two options:

- Create a **new** resource group, and then create a **new** app and App Service plan in the **new** resource group, choosing your desired Azure region during the creation process. You **must** select the **PremiumV2** plan at the time the new app service plan is created. This ensures the combination of resource group, App Service plan, and Azure region will result in the App Service plan being created in an App Service deployment that supports **PremiumV2**. Then redeploy your application code into the newly created app and app service plan. If desired you can subsequently scale the App Service plan down from **PremiumV2** to save costs, and you will still be able to successfully scale back up again in the future using **PremiumV2**.
- If your app already runs in an existing **Premium** tier, then you can clone your app with all app settings, connection strings, and deployment configuration into a new app service plan that uses **PremiumV2**.

app-scaling-premiumv2
App Service

Search (Ctrl+ /)

DEVELOPMENT TOOLS

Clone app

Console

Advanced Tools

App Service Editor (Preview)

Performance test

Click here to access our Quickstart guide for deploying code to your app →

Resource group (change) myResourceGroupWin
Status Running
Location West Europe
Subscription (change) [REDACTED]
Subscription ID [REDACTED]

In the **Clone app** page, you can create an App Service plan using **PremiumV2** in the region you want, and specify the app settings and configuration that you want to clone.

Automate with scripts

You can automate app creation in the **PremiumV2** tier with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

Azure CLI

The following command creates an App Service plan in *P1V2*. You can run it in the Cloud Shell. The options for `--sku` are *P1V2*, *P2V2*, and *P3V2*.

```
az appservice plan create \
--resource-group <resource_group_name> \
--name <app_service_plan_name> \
--sku P1V2
```

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 creates an App Service plan in *P1V2*. The options for `-WorkerSize` are *Small*, *Medium*, and *Large*.

```
New-AzAppServicePlan -ResourceGroupName <resource_group_name> ` 
-Name <app_service_plan_name> ` 
-Location <region_name> ` 
-Tier "PremiumV2" ` 
-WorkerSize "Small"
```

More resources

[Scale up an app in Azure](#)

[Scale instance count manually or automatically](#)

Get started with Autoscale in Azure

12/23/2019 • 3 minutes to read • [Edit Online](#)

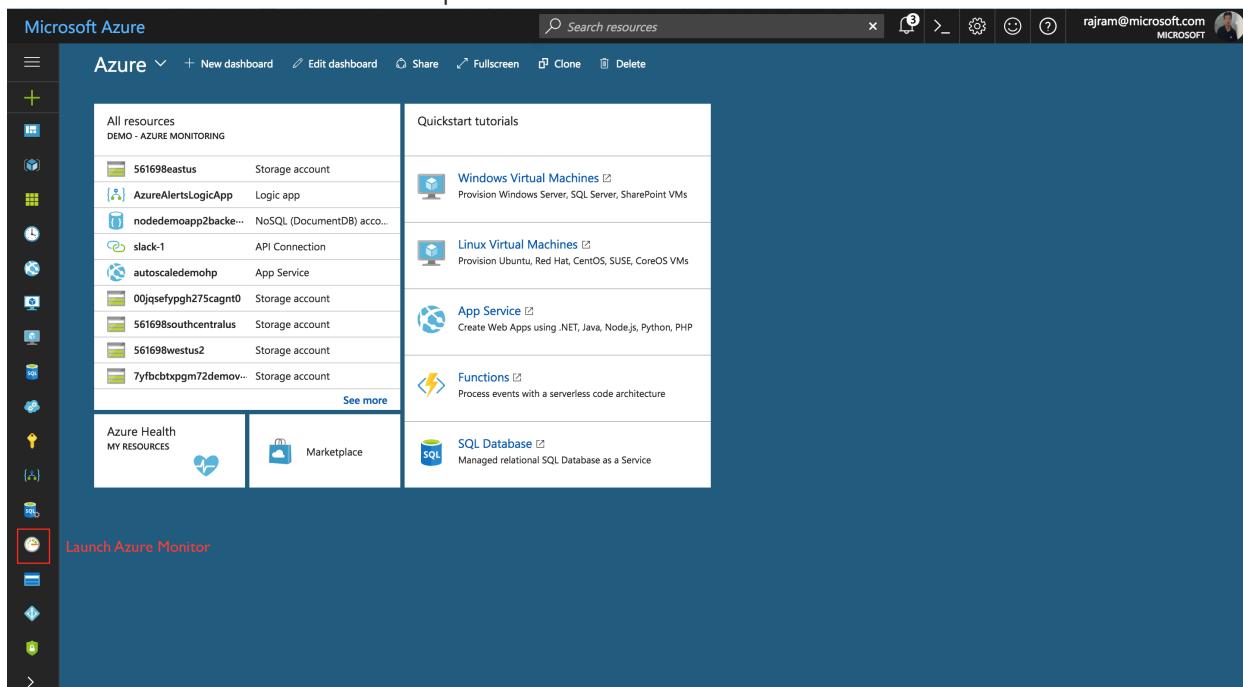
This article describes how to set up your Autoscale settings for your resource in the Microsoft Azure portal.

Azure Monitor autoscale applies only to [Virtual Machine Scale Sets](#), [Cloud Services](#), [App Service - Web Apps](#), and [API Management services](#).

Discover the Autoscale settings in your subscription

You can discover all the resources for which Autoscale is applicable in Azure Monitor. Use the following steps for a step-by-step walkthrough:

1. Open the [Azure portal](#).
2. Click the Azure Monitor icon in the left pane.



3. Click **Autoscale** to view all the resources for which Autoscale is applicable, along with their current Autoscale status.

The screenshot shows the Microsoft Azure Monitor - Autoscale blade. On the left, there's a navigation sidebar with links like 'Activity log', 'Metrics', 'Diagnostics logs', 'Log search', 'Alerts', 'Action groups', and 'Autoscale' (which is highlighted). The main area has a search bar at the top and a table below it. The table columns are: NAME, RESOURCE TYPE, RESOURCE GROUP, LOCATION, INSTANCE COUNT, and AUTOSCALE STATUS. The table lists various resources, including 'WebWorkerDemo' (Cloud service (classic)), 'Production/WebRole1' (Role), 'Production/WorkerRo...' (Role), 'demovmss' (Virtual machine scale set), 'CPUBasedScaleAsp' (App Service plan), 'HolidaySpikeAsp' (App Service plan), 'staticscaleasp' (App Service plan), 'WeekdayTrafficAsp' (App Service plan), 'BrazilSouthPlan' (App Service plan), 'CanadaCentralPlan' (App Service plan), 'SouthCentralUSPlan' (App Service plan), 'WestUS2Plan' (App Service plan), 'contoso-mvc-app-asp' (App Service plan), 'contoso-web-api-asp' (App Service plan), 'contoso-web-react-a...' (App Service plan), and 'nodeedemann2sn' (App Service plan). The 'AUTOSCALE STATUS' column indicates whether Autoscale is 'Not configured' (grey dot), 'Enabled' (green dot), or 'Disabled' (red dot).

You can use the filter pane at the top to scope down the list to select resources in a specific resource group, specific resource types, or a specific resource.

For each resource, you will find the current instance count and the Autoscale status. The Autoscale status can be:

- **Not configured:** You have not enabled Autoscale yet for this resource.
- **Enabled:** You have enabled Autoscale for this resource.
- **Disabled:** You have disabled Autoscale for this resource.

Create your first Autoscale setting

Let's now go through a simple step-by-step walkthrough to create your first Autoscale setting.

1. Open the **Autoscale** blade in Azure Monitor and select a resource that you want to scale. (The following steps use an App Service plan associated with a web app. You can [create your first ASP.NET web app in Azure in 5 minutes](#).)
2. Note that the current instance count is 1. Click **Enable autoscale**.

The screenshot shows the 'Autoscale setting' blade for the 'nodeedemann2sn' App Service plan. At the top, there are 'Save' and 'Discard' buttons, and a link to 'Disable autoscale'. Below that is a toolbar with 'Configure', 'Run history', 'JSON', and 'Notify' buttons. The main area has a section titled 'Override condition' with a slider for 'Instance count' set to 1. A note below says 'Your autoscale configuration is disabled. To reinstate your configuration, enable autoscale.' At the bottom is a blue 'Enable autoscale' button.

3. Provide a name for the scale setting, and then click **Add a rule**. Notice the scale rule options that open as a context pane on the right side. By default, this sets the option to scale your instance count by 1 if the CPU percentage of the resource exceeds 70 percent. Leave it at its default values and click **Add**.

The screenshot shows the 'Autoscale setting' configuration page for the 'nodedemoapp2sp' App Service plan. The 'Scale rule' tab is selected. The 'Default' scale condition is visible, showing 'Scale mode' set to 'Scale based on a metric'. Under 'Rules', there is one rule defined:

- When**: nodedemoapp2sp (Average) CpuPercentage > 70
- Action**: Increase instance count by 1

The 'Criteria' section shows the following settings:

- Metric source**: Current resource (nodedemoapp2sp)
- Resource type**: App Service plans
- Resource**: nodedemoapp2sp
- Criteria**:
 - Time aggregation**: Average
 - Metric name**: CPU Percentage
 - Operator**: Greater than
 - Threshold**: 70
 - Duration (in minutes)**: 10

The 'Action' section shows the operation 'Increase count by 1'.

4. You've now created your first scale rule. Note that the UX recommends best practices and states that "It is recommended to have at least one scale in rule." To do so:

- Click **Add a rule**.
- Set **Operator** to **Less than**.
- Set **Threshold** to **20**.
- Set **Operation** to **Decrease count by**.

You should now have a scale setting that scales out/scales in based on CPU usage.

The screenshot shows the 'Autoscale setting' configuration page for the 'nodedemoapp2sp' App Service plan. The 'Scale rule' tab is selected. The 'Default' scale condition is visible, showing 'Scale mode' set to 'Scale based on a metric'. Under 'Rules', there are two rules defined:

- Scale out**:
 - When**: nodedemoapp2sp (Average) CpuPercentage > 70
 - Action**: Increase instance count by 1
- Scale in**:
 - When**: nodedemoapp2sp (Average) CpuPercentage < 20
 - Action**: Decrease instance count by 1

The 'Criteria' section shows the following settings:

- Metric source**: Current resource (nodedemoapp2sp)
- Resource type**: App Service plans
- Resource**: nodedemoapp2sp
- Criteria**:
 - Time aggregation**: Average
 - Metric name**: CPU Percentage
 - Operator**: Greater than
 - Threshold**: 70
 - Duration (in minutes)**: 10

The 'Action' section shows the operations 'Increase count by 1' and 'Decrease count by 1'.

5. Click **Save**.

Congratulations! You've now successfully created your first scale setting to autoscale your web app based on CPU usage.

NOTE

The same steps are applicable to get started with a virtual machine scale set or cloud service role.

Other considerations

Scale based on a schedule

In addition to scale based on CPU, you can set your scale differently for specific days of the week.

1. Click **Add a scale condition**.
2. Setting the scale mode and the rules is the same as the default condition.
3. Select **Repeat specific days** for the schedule.
4. Select the days and the start/end time for when the scale condition should be applied.

The screenshot shows the Azure portal interface for managing autoscale settings. The top navigation bar includes 'Microsoft Azure', 'Monitor - Autoscale', and 'Autoscale setting'. The main area displays a rule for scaling down when CPU usage is below 20% and increasing instances by 1. Below this, there are fields for minimum (2), maximum (5), and default (2) instance limits. A 'Schedule' section indicates the condition is executed when no other rules match. A red box highlights the 'Auto created scale condition' section, which allows specifying start/end dates and times. In this section, 'Repeat specific days' is selected, and Saturday and Sunday are checked. The start time is set to 00:00 and the end time to 11:59. The 'Timezone' is set to (UTC-08:00) Pacific Time (US & Canada).

Scale differently on specific dates

In addition to scale based on CPU, you can set your scale differently for specific dates.

1. Click **Add a scale condition**.
2. Setting the scale mode and the rules is the same as the default condition.
3. Select **Specify start/end dates** for the schedule.
4. Select the start/end dates and the start/end time for when the scale condition should be applied.

View the scale history of your resource

Whenever your resource is scaled up or down, an event is logged in the activity log. You can view the scale history of your resource for the past 24 hours by switching to the **Run history** tab.

OPERATION NAME	STATUS	EVENT CATEGOR...	TIME	TIME STAMP	SUBSCRIPTION	EVENT INITIATED BY	RESOURCE TYPE	RESOURCE
Scaleup	Succeeded	Autoscale	14 h ago	Sun May 07 2...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	14 h ago	Sat May 06 2...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Sat May 06 2...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	4 d ago	Wed May 03 ...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	4 d ago	Wed May 03 ...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scalein	Cancelled	Autoscale	4 d ago	Wed May 03 ...	Demo - Azure Monitoring	Microsoft.Insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp

If you want to view the complete scale history (for up to 90 days), select [Click here to see more details](#). The activity log opens, with Autoscale pre-selected for your resource and category.

View the scale definition of your resource

Autoscale is an Azure Resource Manager resource. You can view the scale definition in JSON by switching to the **JSON** tab.

```
4   "type": "Microsoft.Insights autoscaleSettings",
5   "location": "West US 2",
6   "tags": {
7     "stype": "Microsoft.WindowsAzure.Management.Common.Storage.CasePreservedDictionary, Microsoft.WindowsAzure.Management.Common.Storage"
8   },
9   "properties": {
10     "profiles": [
11       {
12         "name": "Build conference scale condition",
13         "capacity": {
14           "minimum": "10",
15           "maximum": "10",
16           "default": "10"
17         },
18         "rules": [],
19         "fixedDate": {
20           "timeZone": "Pacific Standard Time",
21           "start": "2017-05-07T00:00:00Z",
22           "end": "2017-05-13T23:59:00Z"
23         }
24       },
25       {
26         "name": "Weekend scale condition",
27         "capacity": {
28           "minimum": "1",
29           "maximum": "1",
30           "default": "1"
31         },
32         "rules": []
33       }
34     ]
35   }
36 }
```

You can make changes in JSON directly, if required. These changes will be reflected after you save them.

Disable Autoscale and manually scale your instances

There might be times when you want to disable your current scale setting and manually scale your resource.

Click the **Disable autoscale** button at the top.

Disable autoscale
Disable your current autoscale configuration? You can reinstate your configuration anytime.

Yes **No**

NOTE

This option disables your configuration. However, you can get back to it after you enable Autoscale again.

You can now set the number of instances that you want to scale to manually.

Override condition

Instance count 4

Your autoscale configuration is disabled. To reinstate your configuration, enable autoscale.

Enable autoscale

You can always return to Autoscale by clicking **Enable autoscale** and then **Save**.

Next steps

- [Create an Activity Log Alert to monitor all Autoscale engine operations on your subscription](#)
- [Create an Activity Log Alert to monitor all failed Autoscale scale-in/scale-out operations on your subscription](#)

High-density hosting on Azure App Service using per-app scaling

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](#)

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
CPU (Short)	The amount of CPU allowed for this app in a 5-minute interval. This quota resets every five minutes.
CPU (Day)	The total amount of CPU allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Memory	The total amount of memory allowed for this app.
Bandwidth	The total amount of outgoing bandwidth allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Filesystem	The total amount of storage allowed.

The only quota applicable to apps that are hosted in *Basic*, *Standard*, and *Premium* is Filesystem.

For more information about the specific quotas, limits, and features available to the various App Service SKUs, see [Azure Subscription service limits](#).

Quota enforcement

If an app exceeds the *CPU (short)*, *CPU (Day)*, or *bandwidth* quota, the app is stopped until the quota resets. During this time, all incoming requests result in an HTTP 403 error.

Error 403 - This web app is stopped.

The web app you have attempted to reach is currently stopped and does not accept any requests. Please try to reload the page or visit it again soon.

If you are the web app administrator, please find the common 403 error scenarios and resolution [here](#). For further troubleshooting tools and recommendations, please visit [Azure Portal](#).

If the app Memory quota is exceeded, the app is 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
Average Response Time	The average time taken for the app to serve requests, in seconds.
Average memory working set	The average amount of memory used by the app, in megabytes (MiB).
Connections	The number of bound sockets existing in the sandbox (w3wp.exe and its child processes). A bound socket is created by calling bind()/connect() APIs and remains until said socket is closed with CloseHandle()/closesocket().
CPU Time	The amount of CPU consumed by the app, in seconds. For more information about this metric, see CPU time vs CPU percentage .
Current Assemblies	The current number of Assemblies loaded across all AppDomains in this application.

METRIC	DESCRIPTION
Data In	The amount of incoming bandwidth consumed by the app, in MiB.
Data Out	The amount of outgoing bandwidth consumed by the app, in MiB.
File System Usage	Percentage of filesystem quota consumed by the app.
Gen 0 Garbage Collections	The number of times the generation 0 objects are garbage collected since the start of the app process. Higher generation GCs include all lower generation GCs.
Gen 1 Garbage Collections	The number of times the generation 1 objects are garbage collected since the start of the app process. Higher generation GCs include all lower generation GCs.
Gen 2 Garbage Collections	The number of times the generation 2 objects are garbage collected since the start of the app process.
Handle Count	The total number of handles currently open by the app process.
Http 2xx	The count of requests resulting in an HTTP status code ≥ 200 but < 300 .
Http 3xx	The count of requests resulting in an HTTP status code ≥ 300 but < 400 .
Http 401	The count of requests resulting in HTTP 401 status code.
Http 403	The count of requests resulting in HTTP 403 status code.
Http 404	The count of requests resulting in HTTP 404 status code.
Http 406	The count of requests resulting in HTTP 406 status code.
Http 4xx	The count of requests resulting in an HTTP status code ≥ 400 but < 500 .
Http Server Errors	The count of requests resulting in an HTTP status code ≥ 500 but < 600 .
IO Other Bytes Per Second	The rate at which the app process is issuing bytes to I/O operations that don't involve data, such as control operations.
IO Other Operations Per Second	The rate at which the app process is issuing I/O operations that aren't read or write operations.
IO Read Bytes Per Second	The rate at which the app process is reading bytes from I/O operations.
IO Read Operations Per Second	The rate at which the app process is issuing read I/O operations.

METRIC	DESCRIPTION
IO Write Bytes Per Second	The rate at which the app process is writing bytes to I/O operations.
IO Write Operations Per Second	The rate at which the app process is issuing write I/O operations.
Memory working set	The current amount of memory used by the app, in MiB.
Private Bytes	Private Bytes is the current size, in bytes, of memory that the app process has allocated that can't be shared with other processes.
Requests	The total number of requests regardless of their resulting HTTP status code.
Requests In Application Queue	The number of requests in the application request queue.
Thread Count	The number of threads currently active in the app process.
Total App Domains	The current number of AppDomains loaded in this application.
Total App Domains Unloaded	The total number of AppDomains unloaded since the start of the application.

For an App Service plan, the available metrics are:

NOTE

App Service plan metrics are available only for plans in *Basic*, *Standard*, and *Premium* tiers.

METRIC	DESCRIPTION
CPU Percentage	The average CPU used across all instances of the plan.
Memory Percentage	The average memory used across all instances of the plan.
Data In	The average incoming bandwidth used across all instances of the plan.
Data Out	The average outgoing bandwidth used across all instances of the plan.
Disk Queue Length	The average number of both read and write requests that were queued on storage. A high disk queue length is an indication of an app that might be slowing down because of excessive disk I/O.
Http Queue Length	The average number of HTTP requests that had to sit on the queue before being fulfilled. A high or increasing HTTP Queue length is a symptom of a plan under heavy load.

CPU time vs CPU percentage

There are two metrics that reflect CPU usage:

CPU Time: Useful for apps hosted in Free or Shared plans, because one of their quotas is defined in CPU minutes used by the app.

CPU percentage: Useful for apps hosted in Basic, Standard, and Premium plans, because they can be scaled out. CPU percentage is a good indication of the overall usage across all instances.

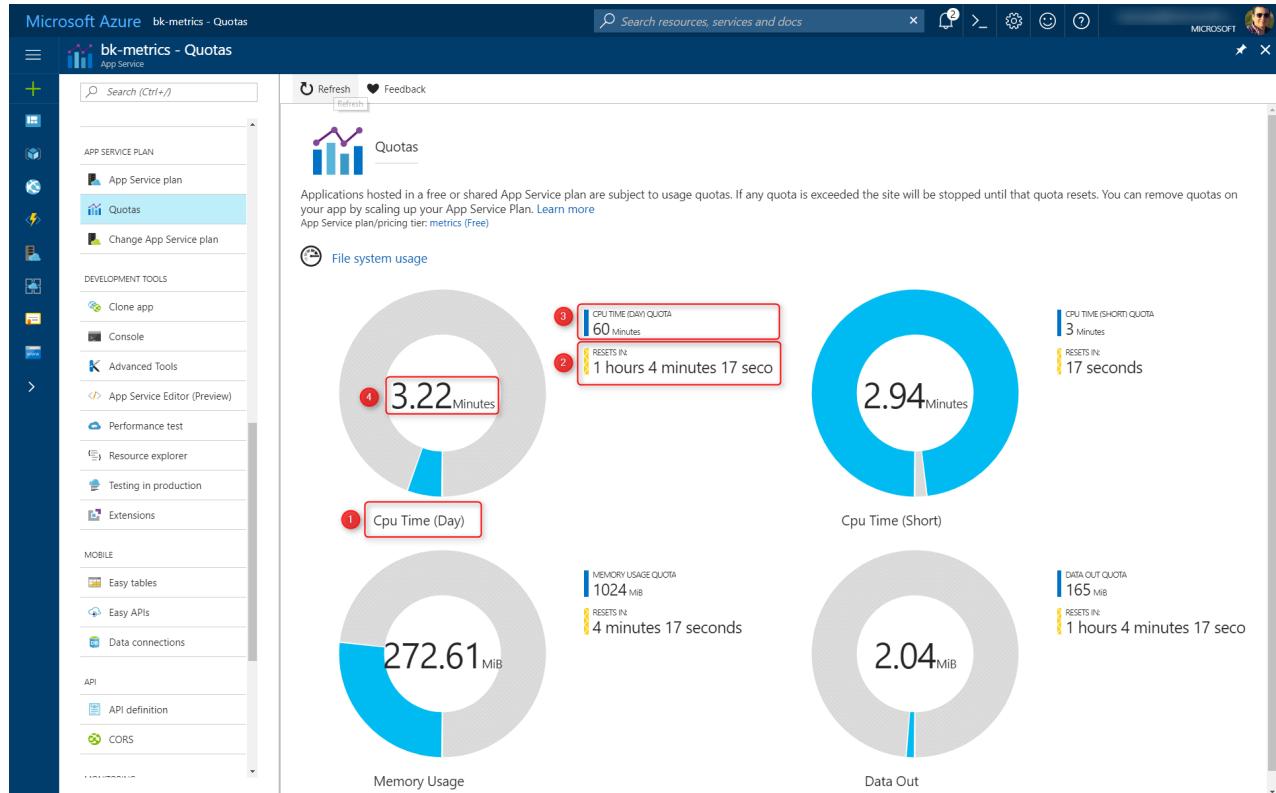
Metrics granularity and retention policy

Metrics for an app and app service plan are logged and aggregated by the service, 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.

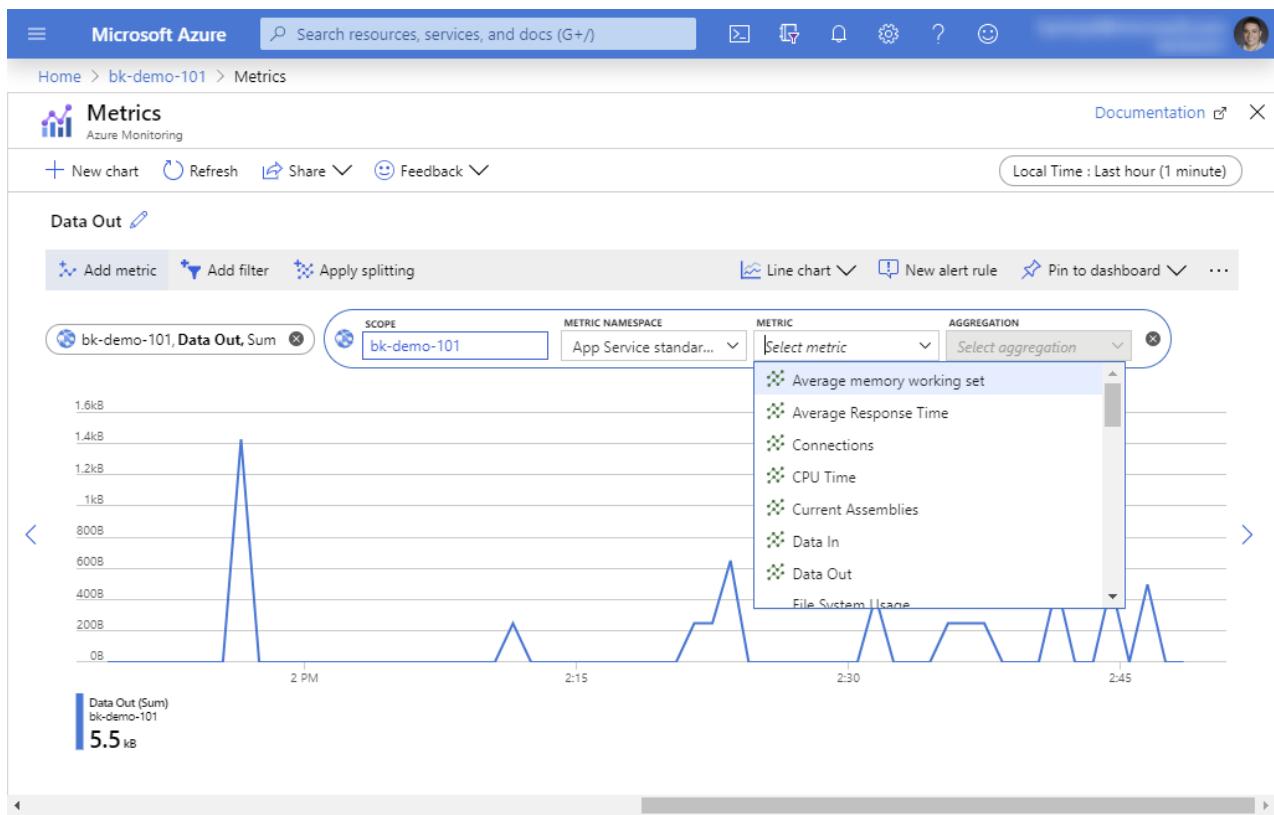
Monitoring quotas and metrics in the Azure portal

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: Critical , Error , Warning , Info , Debug , and Trace . 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 W3C extended log file format . 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 custom deployment script , 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
Disabled	None
Error	Error, Critical
Warning	Warning, Error, Critical
Information	Info, Warning, Error, Critical
Verbose	Trace, Debug, Info, Warning, Error, Critical (all categories)

When finished, select **Save**.

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.Error("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
Application logs	<code>/LogFiles/Application/</code>	Contains one or more text files. The format of the log messages depends on the logging provider you use.
Failed Request Traces	<code>/LogFiles/W3SVC#####/</code>	Contains XML files, and an XSL file. You can view the formatted XML files in the browser.
Detailed Error Logs	<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 Diagnose and solve problems , then search for Failed Request Tracing Logs , then click the icon to browse and view the trace you want.
Web Server Logs	<code>/LogFiles/http/RawLogs/</code>	Contains text files formatted using the W3C extended log file format . This information can be read using a text editor or a utility like Log Parser . App Service doesn't support the <code>s-computername</code> , <code>s-ip</code> , or <code>cs-version</code> fields.

LOG TYPE	DIRECTORY	DESCRIPTION
Deployment logs	/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 App Service Editor (Preview) interface. 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 in the API section is highlighted with a red box. The main area shows 'Subscription * ⓘ' set to 'Demo Two Subscription' and 'Resource group ⓘ' set to 'appsvc-azmon-rg'. Below that, 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 message says 'No diagnostic settings defined' and there's a red box around the '+ Add diagnostic setting' button. Below this, 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](#)

Manage an App Service plan in Azure

1/6/2020 • 3 minutes to read • [Edit Online](#)

An [Azure App Service plan](#) provides the resources that an App Service app needs to run. This guide shows how to manage an App Service plan.

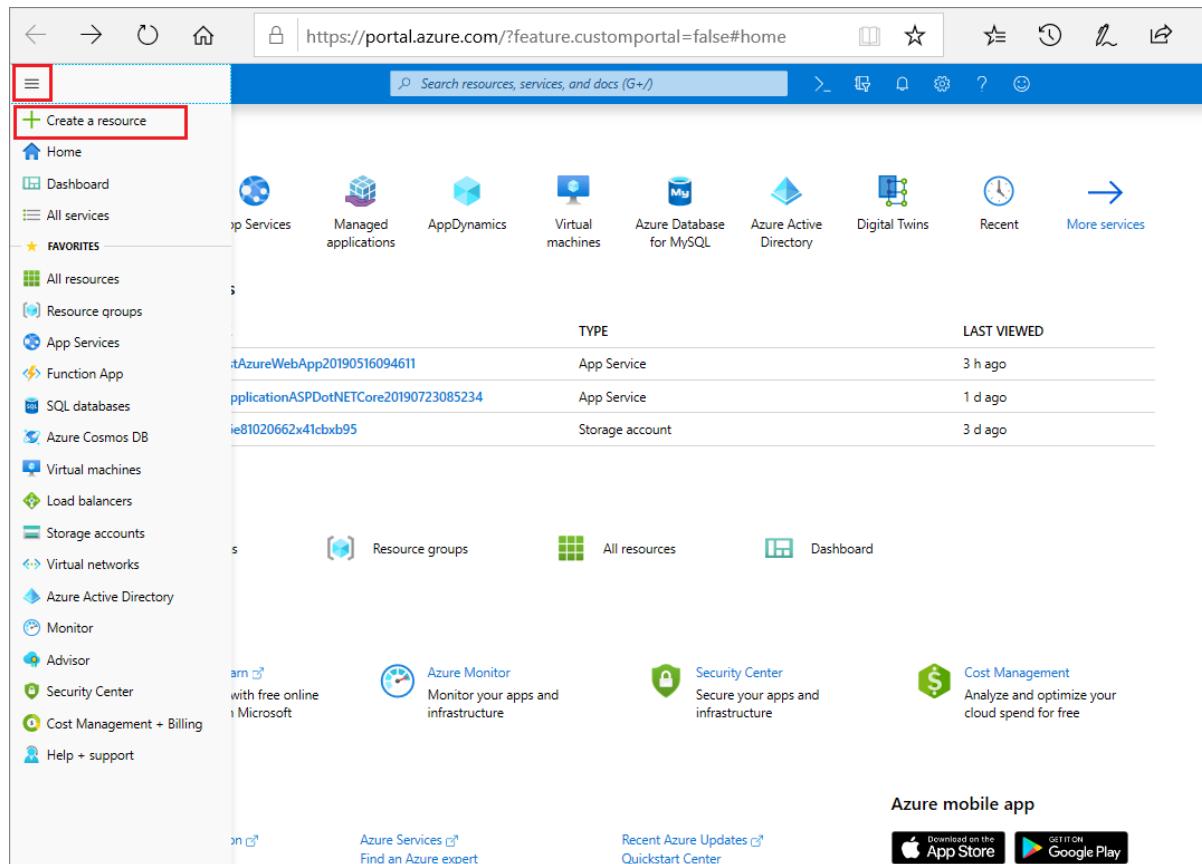
Create an App Service plan

TIP

If you have an App Service Environment, see [Create an App Service plan in an App Service Environment](#).

You can create an empty App Service plan, or you can create a plan as part of app creation.

1. In the [Azure portal](#), select **Create a resource**.



The screenshot shows the Azure portal homepage. On the left, there's a navigation sidebar with various service icons and links like Home, Dashboard, All services, Favorites, All resources, Resource groups, App Services, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Cost Management + Billing, and Help + support. A red box highlights the 'Create a resource' button at the top of the sidebar. The main content area has a search bar at the top. Below it, there are several service tiles: App Services, Managed applications, AppDynamics, Virtual machines, Azure Database for MySQL, Azure Active Directory, Digital Twins, and a Recent services section. Below these is a table showing recently viewed resources: 'tAzureWebApp20190516094611' (App Service, 3 h ago), 'applicationASPDotNETCore20190723085234' (App Service, 1 d ago), and 'ie81020662x41cbxb95' (Storage account, 3 d ago). At the bottom, there are links for Azure mobile app, Azure Services, Recent Azure Updates, and Quickstart Center, along with download links for the App Store and Google Play.

2. Select **New > Web App** or another kind of App service app.

The screenshot shows the Azure Marketplace interface. At the top left, there's a breadcrumb navigation: Home > New. Below it, a search bar contains the placeholder text "Search the Marketplace". On the left, a sidebar lists various service categories: Get started, Recently created, AI + Machine Learning, Analytics, Blockchain, Compute, Containers, Databases, Developer Tools, DevOps, Identity, Integration, and Internet of Things. The "Web App" option is highlighted with a red box. To its right, under the "Popular" heading, are cards for Windows Server 2016 Datacenter, Ubuntu Server 18.04 LTS, SQL Database, Function App, and Azure Cosmos DB, each with a "Quickstart tutorial" link.

3. Configure the **Instance Details** section before configuring the App Service plan. Settings such as **Publish** and **Operating Systems** can change the available pricing tiers for your App Service plan. **Region** determines where your App Service plan is created.
4. In the **App Service Plan** section, select an existing plan, or create a plan by selecting **Create new**.

The screenshot shows the "Web App" creation wizard. The title bar says "Web App". The first section is "Region *", which has a dropdown set to "Central US" with a note below it: "Can't find your App Service Plan, try a different region.". The second section is "App Service Plan". It shows a dropdown for "Linux Plan (Central US) * ①" with "(New) ASP-myResourceGroup-bfdd" selected and a "Create new" button below it. To the right, there's a "Sku and size *" section with "Premium V2 P1v2" selected, showing "210 total ACU, 3.5 GB memory" and a "Change size" link. At the bottom, there are three buttons: "Review + create" (highlighted with a red box), "< Previous", and "Next : Tags >".

5. When creating a plan, you can select the pricing tier of the new plan. In **Sku and size**, select **Change size** to change the pricing tier.

Move an app to another App Service plan

You can move an app to another App Service plan, as long as the source plan and the target plan are in the *same resource group and geographical region*.

NOTE

Azure deploys each new App Service plan into a deployment unit, internally called a webspace. Each region can have many webspaces, but your app can only move between plans that are created in the same webspace. An App Service Environment is an isolated webspace, so apps can be moved between plans in the same App Service Environment, but not between plans in different App Service Environments.

You can't specify the webspace you want when creating a plan, but it's possible to ensure that a plan is created in the same webspace as an existing plan. In brief, all plans created with the same resource group and region combination are deployed into the same webspace. For example, if you created a plan in resource group A and region B, then any plan you subsequently create in resource group A and region B is deployed into the same webspace. Note that plans can't move webspaces after they're created, so you can't move a plan into "the same webspace" as another plan by moving it to another resource group.

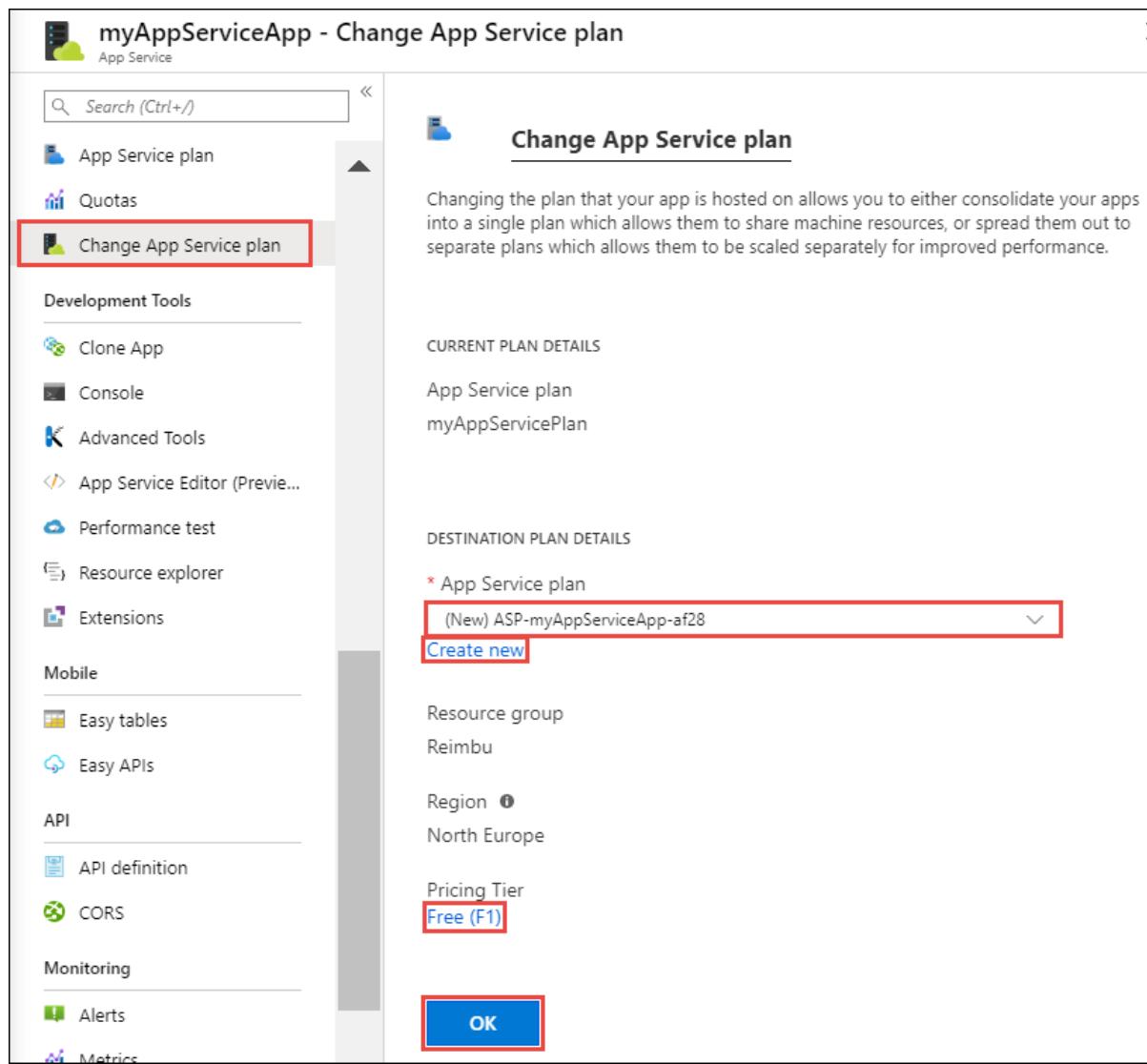
1. In the [Azure portal](#), search for and select **App services** and select the app that you want to move.
2. From the left menu, select **Change App Service plan**.
3. In the **App Service plan** dropdown, select an existing plan to move the app to. The dropdown shows only plans that are in the same resource group and geographical region as the current App Service plan. If no such plan exists, it lets you create a plan by default. You can also create a new plan manually by selecting **Create new**.
4. If you create a plan, you can select the pricing tier of the new plan. In **Pricing Tier**, select the existing tier to change it.

IMPORTANT

If you're moving an app from a higher-tiered plan to a lower-tiered plan, such as from **D1** to **F1**, the app may lose certain capabilities in the target plan. For example, if your app uses SSL certificates, you might see this error message:

```
Cannot update the site with hostname '<app_name>' because its current SSL configuration 'SNI based SSL enabled' is not allowed in the target compute mode. Allowed SSL configuration is 'Disabled'.
```

5. When finished, select **OK**.



Move an app to a different region

The region in which your app runs is the region of the App Service plan it's in. However, you cannot change an App Service plan's region. If you want to run your app in a different region, one alternative is app cloning. Cloning makes a copy of your app in a new or existing App Service plan in any region.

You can find **Clone App** in the **Development Tools** section of the menu.

IMPORTANT

Cloning has some limitations. You can read about them in [Azure App Service App cloning](#).

Scale an App Service plan

To scale up an App Service plan's pricing tier, see [Scale up an app in Azure](#).

To scale out an app's instance count, see [Scale instance count manually or automatically](#).

Delete an App Service plan

To avoid unexpected charges, when you delete the last app in an App Service plan, App Service also deletes the plan by default. If you choose to keep the plan instead, you should change the plan to **Free** tier so you're not charged.

IMPORTANT

App Service plans that have no apps associated with them still incur charges because they continue to reserve the configured VM instances.

Next steps

[Scale up an app in Azure](#)

Back up your app in Azure

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 Azure portal interface for an 'App Service'. The top navigation bar has the name 'contoso-backup'. Below it is a search bar with placeholder text 'Search (Ctrl+ /)'. The main area is titled 'SETTINGS' and contains several options: 'Application settings', 'Authentication / Authorization', 'Backups' (which is highlighted with a red box), 'Custom domains', 'SSL certificates', 'Networking', and 'Scale up (App Service plan)'.

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](#).



Backup and Restore feature requires your App Service plan to be Standard or higher. Click here to upgrade your App Service plan and access this feature.

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, there's a link to 'Configure backup to create restorable archive copies of your apps content, configuration and database.' followed by a 'Learn more' link. A prominent message 'Backup is not configured. Click here to configure backup for your app.' is displayed in a box with a gear icon, and this entire message box is highlighted with a red border.

3. In the **Backup Configuration** page, click **Storage not configured** to configure a storage account.

The screenshot shows the 'Backup Storage' configuration page. It includes a 'Backup Storage' icon and the text 'Select the target container to store your app backup.'. Below this is a 'Storage Settings' section containing the message 'Storage not configured' and a right-pointing arrow. This entire 'Storage Settings' section is highlighted with a red border.

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.

<input type="checkbox"/> 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 (UTC+02:00) --- Current Time Zone ---

* 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 Azure portal interface for the 'contoso-backup' app. The left sidebar lists various settings like Overview, Activity log, and Deployment slots. The 'Backups' option is highlighted with a red box. The main content area is titled 'Backup' and contains a status message: 'Backup configured, Schedule for backup is not configured, Configure scheduled backup to automatically take backups.' Below this are two large buttons: 'Configure' (blue) and 'Restore' (red, also highlighted with a red box). A table at the bottom shows a single backup entry: a green checkmark icon, 'S...', 'Thursday, June 8, 2017..', and '0.06 MB'. The table has columns for STATUS, BACKUP TIME, and SIZE (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 menu. Under 'Resource group', it shows 'cephalin-appwithsql'. Under 'Status', it says 'Primary: Available'. Under 'Location', it shows 'West Europe'. Under 'Subscription name', it shows 'Visual Studio Ultimate with MSDN'. There is also a 'Subscription ID' field with a redacted value. To the right of these details are icons for Performance/Access tier (Standard/Hot), Replication (Locally-redundant storage (LRS)), and Blob service endpoint (https://cephalinstorage4.blob.core.windows..). A 'Pencil' icon is next to the subscription name. At the bottom of the essentials section is a link 'All settings →'. Below this is a search bar with placeholder text 'Search containers by prefix'. The main table lists one container named 'backups' with its URL and last modified date (7/6/2016, 2:00:16 PM). A red box highlights the first column of this row.

NAME	URL	LAST MODIFIED
backups	https://cephalinstorage4.blo...	7/6/2016, 2:00:16 PM

4. Select backup file you want to download or delete.

NAME	MODIFIED	BLOB TYPE	SIZE
cephalin-appwithsql_201607061211.log	7/6/2016, 2:15:58..	Block blob	272 B
cephalin-appwithsql_201607061211.xml	7/6/2016, 2:15:58..	Block blob	793 B
cephalin-appwithsql_201607061211.zip	7/6/2016, 2:15:58..	Block blob	151.11 KB

5. Click **Download** or **Delete** depending on what you want to do.

Monitor a restore operation

To see details about the success or failure of the app restore operation, navigate to the **Activity Log** page in the Azure portal.

Scroll down to find the desired restore operation and click to select it.

The details page displays the available information related to the restore operation.

Automate with scripts

You can automate backup management with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

For samples, see:

- [Azure CLI samples](#)
- [Azure PowerShell samples](#)

Restore an app in Azure from a snapshot

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

This article shows you how to restore an app in [Azure App Service](#) from a snapshot. You can restore your app to a previous state, based on one of your app's snapshots. You do not need to enable snapshots backup, the platform automatically saves a snapshot of all apps for data recovery purposes.

Snapshots are incremental shadow copies, and they offer several advantages over regular [backups](#):

- No file copy errors due to file locks.
- No storage size limitation.
- No configuration required.

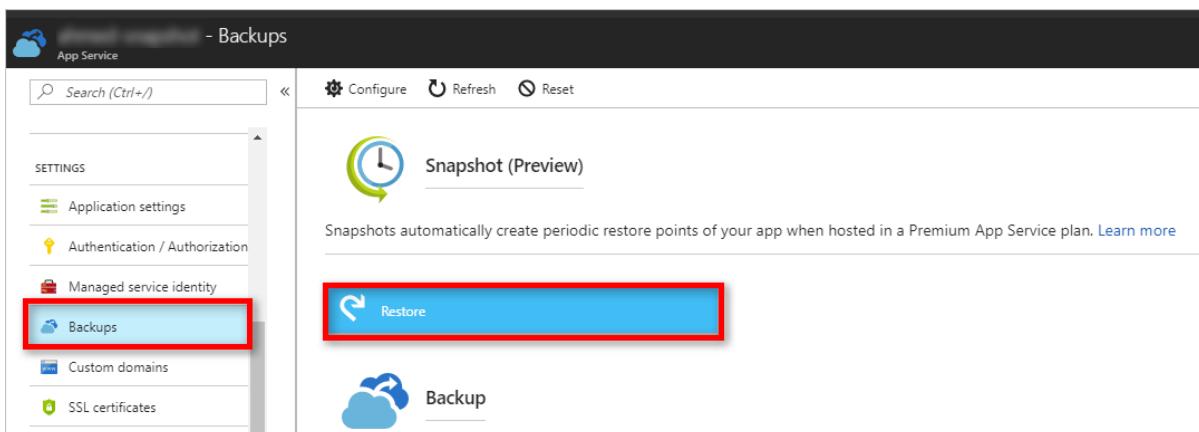
Restoring from snapshots is available to apps running in **Premium** tier or higher. For information about scaling up your app, see [Scale up an app in Azure](#).

Limitations

- The feature is currently in preview.
- You can only restore to the same app or to a slot belonging to that app.
- App Service stops the target app or target slot while doing the restore.
- App Service keeps three months worth of snapshots for platform data recovery purposes.
- You can only restore snapshots for the last 30 days.
- App Services running on an App Service Environment do not support snapshots.

Restore an app from a snapshot

1. On the **Settings** page of your app in the [Azure portal](#), click **Backups** to display the **Backups** page. Then click **Restore** under the **Snapshot(Preview)** section.



2. In the **Restore** page, select the snapshot to restore.



Select Backup to Restore

Select from either a Backup on the app or zip file of a valid backup from a storage container.

Restore source ⓘ App backup Storage Snapshot (Preview)

Snapshot (Preview) ⓘ

Snapshot taken at Monday, October 23, 2017, 10:41:49 PM PDT

3. Specify the destination for the app restore in **Restore destination**.



Select a target App Service App

Select to overwrite 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 app's current file system is erased and overwritten. Before you click **OK**, make sure that it is what you want to do.

NOTE

Due to current technical limitations, you can only restore to apps in the same scale unit. This limitation will be removed in a future release.

You can select **Existing App** to restore to a slot. Before you use this option, you should have already created a slot in your app.

4. You can choose to restore your site configuration.



Advanced Settings

Advanced settings for restoring an app backup with options.

Restore site configuration No Yes

5. Click **OK**.

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](#)

Restore deleted App Service app Using PowerShell

1/6/2020 • 2 minutes to read • [Edit Online](#)

If you happened to accidentally delete your app in Azure App Service, you can restore it using the commands from the [Az PowerShell module](#).

NOTE

Deleted apps are purged from the system 30 days after the initial deletion. Once an app has been purged, it can't be recovered.

Re-register App Service resource provider

Some customers might come across an issue where retrieving the list of deleted apps fails. To resolve the issue, run the following command:

```
Register-AzResourceProvider -ProviderNamespace "Microsoft.Web"
```

List deleted apps

To get the collection of deleted apps, you can use `Get-AzDeletedWebApp`.

For details on a specific deleted app you can use:

```
Get-AzDeletedWebApp -Name <your_deleted_app> -Location <your_deleted_app_location>
```

The detailed information includes:

- **DeletedSiteId**: Unique identifier for the app, used for scenarios where multiple apps with the same name have been deleted
- **SubscriptionID**: Subscription containing the deleted resource
- **Location**: Location of the original app
- **ResourceGroupName**: Name of the original resource group
- **Name**: Name of the original app.
- **Slot**: the name of the slot.
- **Deletion Time**: When was the app deleted

Restore deleted app

Once the app you want to restore has been identified, you can restore it using `Restore-AzDeletedWebApp`.

```
Restore-AzDeletedWebApp -ResourceGroupName <my_rg> -Name <my_app> -TargetAppServicePlanName <my_asp>
```

The inputs for command are:

- **Resource Group**: Target resource group where the app will be restored
- **Name**: Name for the app, should be globally unique.

- **TargetAppServicePlanName:** App Service plan linked to the app

By default `Restore-AzDeletedWebApp` will restore both your app configuration as well a content. If you want to only restore content, you use the `-RestoreContentOnly` flag with this commandlet.

NOTE

If the app was hosted on and then deleted from an App Service Environment then it can be restored only if the corresponding App Service Environment still exist.

You can find the full commandlet reference here: [Restore-AzDeletedWebApp](#).

Move an App Service app to another region

2/28/2020 • 3 minutes to read • [Edit Online](#)

This article describes how to move App Service resources to a different Azure region. You might move your resources to another region for a number of reasons. For example, to take advantage of a new Azure region, to deploy features or services available in specific regions only, to meet internal policy and governance requirements, or in response to capacity planning requirements.

App Service resources are region-specific and can't be moved across regions. You must create a copy of your existing App Service resources in the target region, move your content over to the new app. If your source app uses a custom domain, you can [migrate it to the new app in the target region](#) when you're finished.

To make copying your app easier, you can [clone an individual App Service app](#) into an App Service plan in another region, but it does have [limitations](#), especially that it doesn't support Linux apps.

Prerequisites

- Make sure that the App Service app is in the Azure region from which you want to move.
- Make sure that the target region supports App Service and any related service, whose resources you want to move.

Prepare

Identify all the App Service resources that you're currently using. For example:

- App Service apps
- [App Service plans](#)
- [Deployment slots](#)
- [Custom domains purchased in Azure](#)
- [SSL certificates](#)
- [Azure Virtual Network integration](#)
- [Hybrid connections.](#)
- [Managed identities](#)
- [Backup settings](#)

Certain resources, such as imported certificates or hybrid connections, contain integration with other Azure services. For information on how to move those resources across regions, see the documentation for the respective services.

Move

1. [Create a back up of the source app.](#)
2. [Create an app in a new App Service plan, in the target region.](#)
3. [Restore the back up in the target app](#)
4. If you use a custom domain, [bind it preemptively to the target app](#) with `awverify.` and [enable the domain in the target app.](#)
5. Configure everything else in your target app to be the same as the source app and verify your configuration.
6. When you're ready for the custom domain to point to the target app, [remap the domain name.](#)

Clean up source resources

Delete the source app and App Service plan. [An App Service plan in the non-free tier carries a charge, even if no app is running in it.](#)

Next steps

[Azure App Service App Cloning Using PowerShell](#)

Move resources to a new resource group or subscription

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:

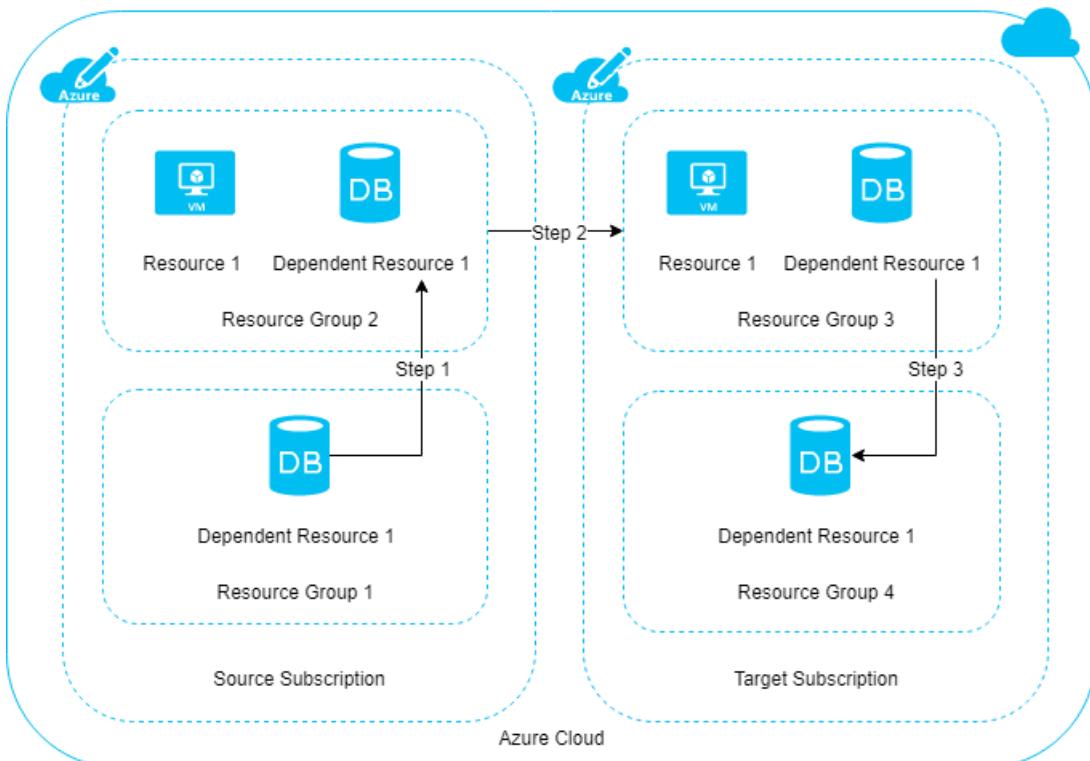
- **Microsoft.Resources/subscriptions/resourceGroups/moveResources/action** on the source resource group.
 - **Microsoft.Resources/subscriptions/resourceGroups/write** on the destination resource group.
7. Before moving the resources, check the subscription quotas for the subscription you're moving the 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:

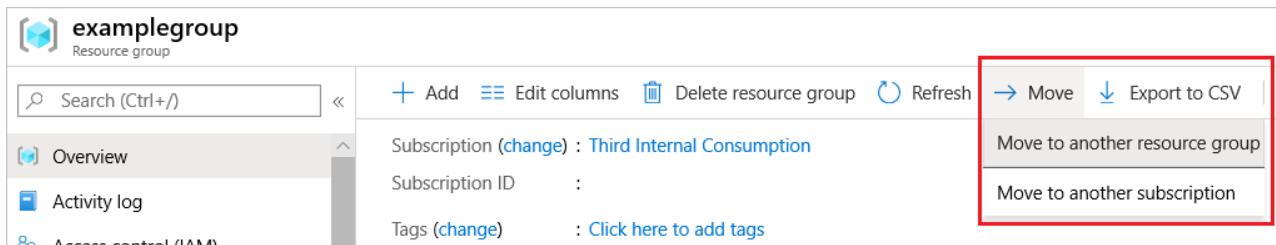
```
GET <location-url>
Authorization: Bearer <access-token>
```

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 Resource Groups blade for a resource group named "examplegroup". On the right side, there is a toolbar with several buttons: "Add", "Edit columns", "Delete resource group", "Refresh", "Move", and "Export to CSV". The "Move" button is highlighted with a red box. Below the toolbar, there is some descriptive text: "Subscription (change) : Third Internal Consumption", "Subscription ID : [redacted]", and "Tags (change) : Click here to add tags".

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		Type
<input checked="" type="checkbox"/>	Select all	
<input checked="" type="checkbox"/>	 Failure Anomalies - tfexamplesite	microsoft.alertsmanagement/smarterdetectoralertrules
<input checked="" type="checkbox"/>	 Application Insights Smart Detection	microsoft.insights/actiongroups
<input checked="" type="checkbox"/>	 tfexamplesite	Application Insights
<input checked="" type="checkbox"/>	 ASP-examplegroup-a26c	App Service plan
<input checked="" type="checkbox"/>	 tfexamplesite	App Service

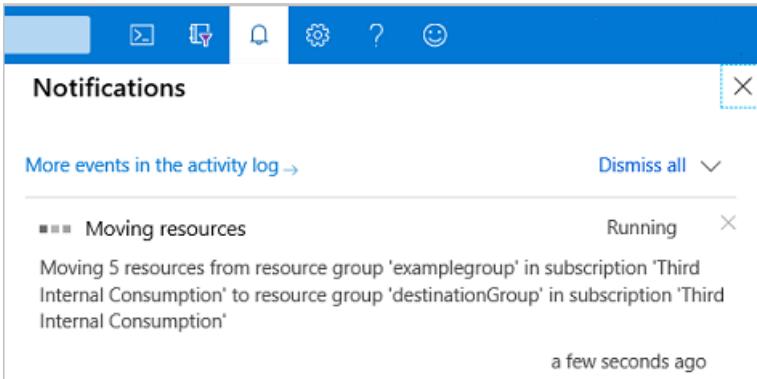
Move these resources to

Resource group *

I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs (i)

OK

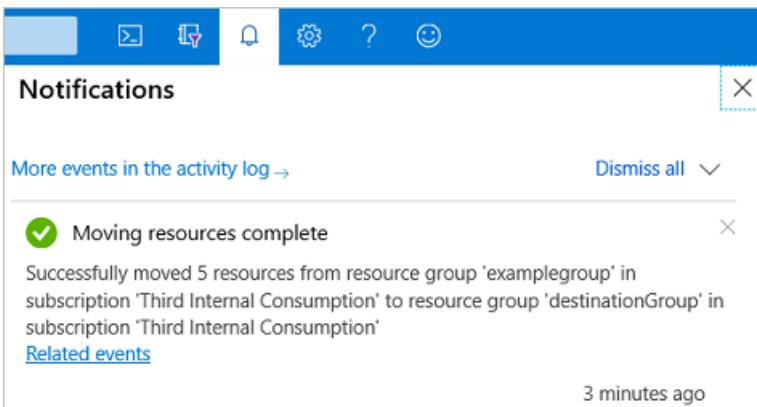
In **Notifications**, you see that the move operation is running.



The Notifications panel shows a single event:

- Moving resources**: Status is **Running**.
- Description: Moving 5 resources from resource group 'examplegroup' in subscription 'Third Internal Consumption' to resource group 'destinationGroup' in subscription 'Third Internal Consumption'.
- Timestamp: a few seconds ago.

When it has completed, you're notified of the result.



The Notifications panel shows a completed event:

- Moving resources complete**: Status is **Success**.
- Description: Successfully moved 5 resources from resource group 'examplegroup' in subscription 'Third Internal Consumption' to resource group 'destinationGroup' in subscription 'Third Internal Consumption'.
- [Related events](#)
- Timestamp: 3 minutes ago.

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.

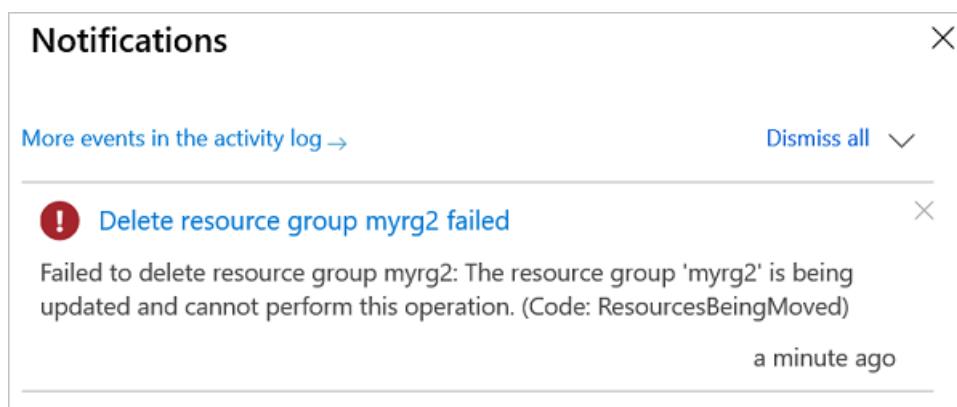
If a resource can't be moved within the 4-hour window, Resource Manager unlocks both resource groups.

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](#).

Run background tasks with WebJobs in Azure App Service

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

This article shows how to deploy WebJobs by using the [Azure portal](#) to upload an executable or script. For information about how to develop and deploy WebJobs by using Visual Studio, see [Deploy WebJobs using Visual Studio](#).

Overview

WebJobs is a feature of [Azure App Service](#) that enables you to run a program or script in the same context as a web app, API app, or mobile app. There is no additional cost to use WebJobs.

IMPORTANT

WebJobs is not yet supported for App Service on Linux.

The Azure WebJobs SDK can be used with WebJobs to simplify many programming tasks. For more information, see [What is the WebJobs SDK](#).

Azure Functions provides another way to run programs and scripts. For a comparison between WebJobs and Functions, see [Choose between Flow, Logic Apps, Functions, and WebJobs](#).

WebJob types

The following table describes the differences between *continuous* and *triggered* WebJobs.

CONTINUOUS	TRIGGERED
Starts immediately when the WebJob is created. To keep the job from ending, the program or script typically does its work inside an endless loop. If the job does end, you can restart it.	Starts only when triggered manually or on a schedule.
Runs on all instances that the web app runs on. You can optionally restrict the WebJob to a single instance.	Runs on a single instance that Azure selects for load balancing.
Supports remote debugging.	Doesn't support remote debugging.

NOTE

A web app can time out after 20 minutes of inactivity. Only requests to the actual web app reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (https://<app_name>.scm.azurewebsites.net) don't reset the timer.

If your app runs continuous or scheduled (Timer trigger) WebJobs, enable **Always On** to ensure that the WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium [pricing tiers](#).

Supported file types for scripts or programs

The following file types are supported:

- .cmd, .bat, .exe (using Windows cmd)
- .ps1 (using PowerShell)
- .sh (using Bash)
- .php (using PHP)
- .py (using Python)
- .js (using Node.js)
- .jar (using Java)

Create a continuous WebJob

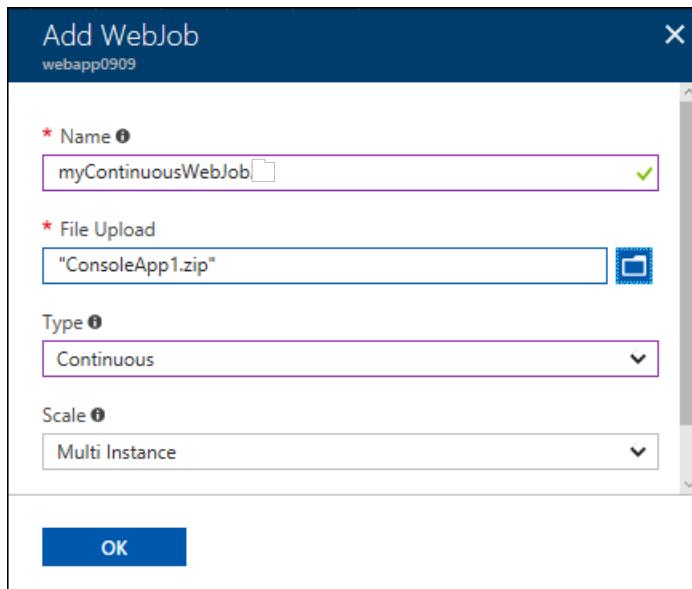
1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. Select **WebJobs**.

The screenshot shows the Azure portal interface for an App Service named 'WebApp0909'. The left sidebar lists various configuration options: Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' option is highlighted with a red box.

3. In the **WebJobs** page, select **Add**.

The screenshot shows the 'WebJobs' page for the 'WebApp0909' app service. The top navigation bar includes a search bar, a 'Add' button (highlighted with a red box), a 'Refresh' button, 'Logs', 'Delete', and 'Properties' buttons. The left sidebar lists options: Push, MySQL In App, Properties, Locks, and Automation script. The main area displays a 'WebJobs' section with a gear icon, a brief description, and a table header for 'NAME', 'TYPE', 'STATUS', and 'SCHEDULE'. A message at the bottom states: 'You haven't added any WebJobs. Click ADD to get started.'

4. Use the **Add WebJob** settings as specified in the table.



SETTING	SAMPLE VALUE	DESCRIPTION
Name	myContinuousWebJob	A name that is unique within an App Service app. Must start with a letter or a number and cannot contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A .zip file that contains your executable or script file as well as any supporting files needed to run the program or script. The supported executable or script file types are listed in the Supported file types section.
Type	Continuous	The WebJob types are described earlier in this article.
Scale	Multi instance	Available only for Continuous WebJobs. Determines whether the program or script runs on all instances or just one instance. The option to run on multiple instances doesn't apply to the Free or Shared pricing tiers.

5. Click **OK**.

The new WebJob appears on the **WebJobs** page.

The screenshot shows the Azure portal interface for an App Service named "WebApp0909". On the left, a sidebar titled "SETTINGS" lists various configuration options: Application settings, Authentication / Authorization (highlighted in yellow), Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The "WebJobs" option is selected and highlighted in blue. The main content area is titled "WebJobs" and contains a brief description: "WebJobs provide an easy way to run scripts or programs as background processes". Below this is a table listing three WebJobs:

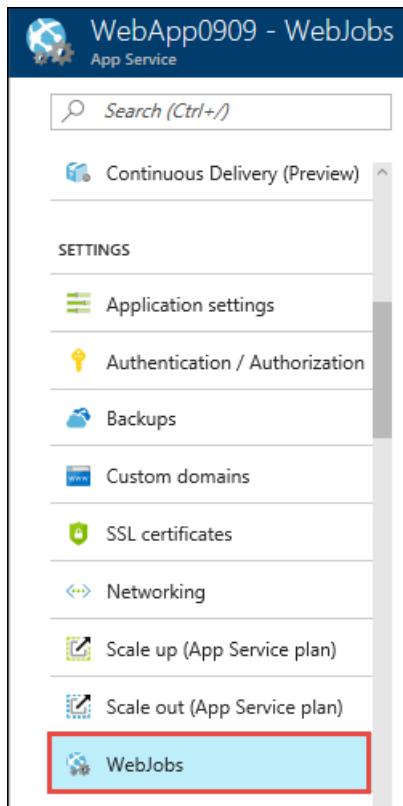
NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Ready	0 0/20 * * *
myTriggeredWeb...	Triggered	Ready	n/a
myContinuousW...	Continuous	Pending Restart	n/a

6. To stop or restart a continuous WebJob, right-click the WebJob in the list and click **Stop** or **Start**.

This screenshot shows the same "WebJobs" list as the previous one. A context menu is open over the "myContinuousW..." row. The menu items are: Logs (with a file icon), Delete (with a trash bin icon), Stop (with a square icon), and Properties (with a gear icon). The "Stop" item is highlighted with a red box.

Create a manually triggered WebJob

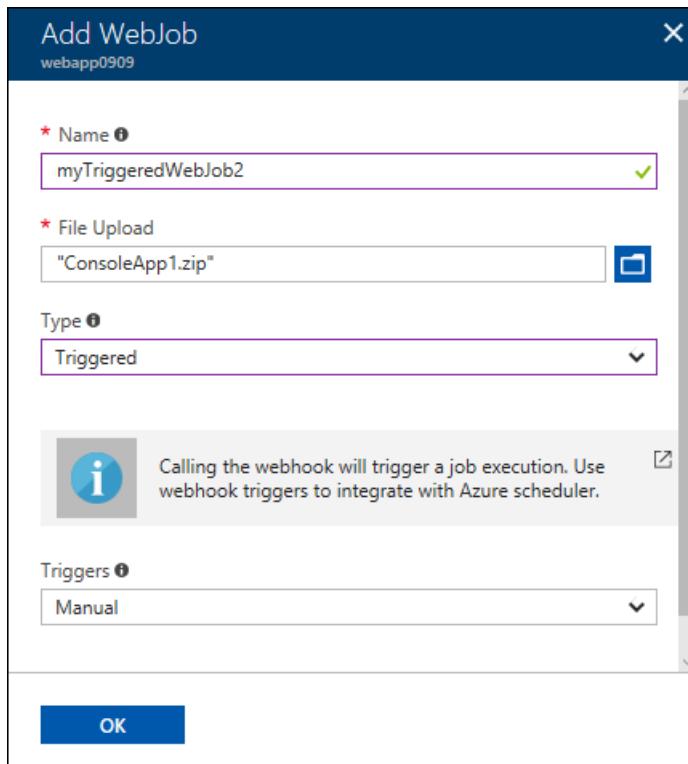
1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. Select **WebJobs**.



3. In the **WebJobs** page, select **Add**.

A screenshot of the 'WebJobs' add page. The left sidebar lists options: WebJobs (highlighted with a blue box), Push, MySQL In App, Properties, Locks, and Automation script. The main area has a title 'WebJobs' with a gear icon. Below it, a message says 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' A table header with columns NAME, TYPE, STATUS, and SCHEDULE is shown, followed by the message 'You haven't added any WebJobs. Click ADD to get started.' A red box highlights the '+ Add' button in the top navigation bar.

4. Use the **Add WebJob** settings as specified in the table.



SETTING	SAMPLE VALUE	DESCRIPTION
Name	myTriggeredWebJob	A name that is unique within an App Service app. Must start with a letter or a number and cannot contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A .zip file that contains your executable or script file as well as any supporting files needed to run the program or script. The supported executable or script file types are listed in the Supported file types section.
Type	Triggered	The WebJob types are described earlier in this article.
Triggers	Manual	

5. Click **OK**.

The new WebJob appears on the **WebJobs** page.

WebApp0909 - WebJobs

App Service

Search (Ctrl+ /)

SETTINGS

- Application settings
- Authentication / Authorization
- Backups
- Custom domains
- SSL certificates
- Networking
- Scale up (App Service plan)
- Scale out (App Service plan)
- WebJobs

WebJobs

WebJobs provide an easy way to run scripts or programs as background processes

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Ready	0 0/20 * * *
myTriggeredWeb...	Triggered	Ready	n/a
myContinuousW...	Continuous	Pending Restart	n/a

6. To run the WebJob, right-click its name in the list and click **Run**.

+

Add Refresh Logs Delete Properties

WebJobs

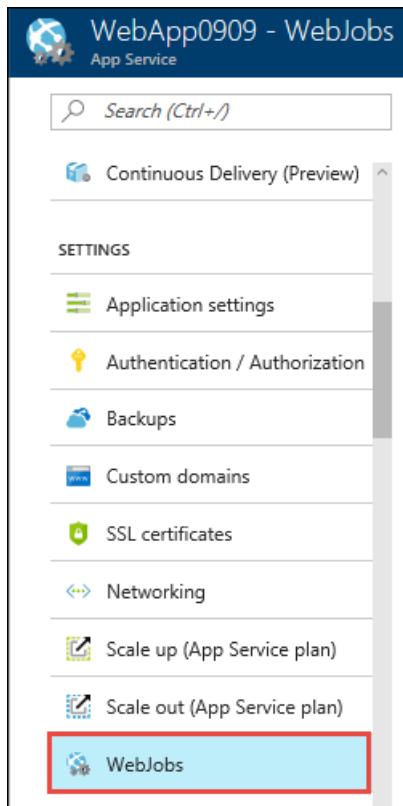
WebJobs provide an easy way to run scripts or programs as background processes

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Ready	0 0/20 * * *
myTriggeredWeb...	Triggered	Ready	n/a
myContinuousW...	Continuous	n/a	

Run

Create a scheduled WebJob

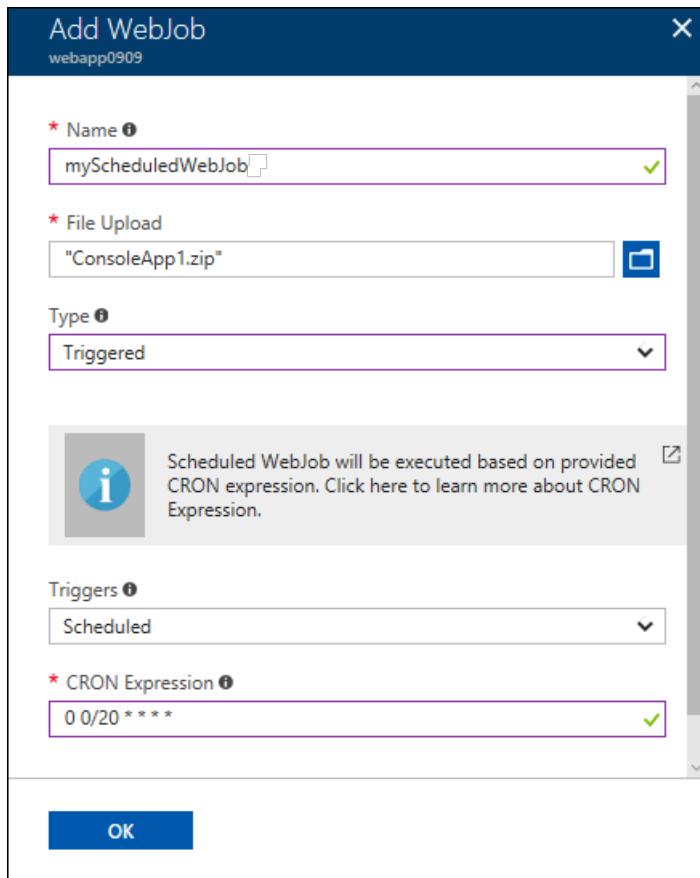
1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. Select **WebJobs**.



3. In the **WebJobs** page, select **Add**.

A screenshot of the 'WebJobs' add page. The left sidebar lists options: WebJobs (highlighted with a blue box), Push, MySQL In App, Properties, Locks, and Automation script. The main area has a 'WebJobs' icon and the text 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' Below is a table with columns NAME, TYPE, STATUS, and SCHEDULE. A message says 'You haven't added any WebJobs. Click ADD to get started.' The 'Add' button in the top toolbar is highlighted with a red box.

4. Use the **Add WebJob** settings as specified in the table.



SETTING	SAMPLE VALUE	DESCRIPTION
Name	myScheduledWebJob	A name that is unique within an App Service app. Must start with a letter or a number and cannot contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A .zip file that contains your executable or script file as well as any supporting files needed to run the program or script. The supported executable or script file types are listed in the Supported file types section.
Type	Triggered	The WebJob types are described earlier in this article.
Triggers	Scheduled	For the scheduling to work reliably, enable the Always On feature. Always On is available only in the Basic, Standard, and Premium pricing tiers.
CRON Expression	0 0/20 * * *	CRON expressions are described in the following section.

5. Click **OK**.

The new WebJob appears on the **WebJobs** page.

The screenshot shows the Azure portal interface for managing a WebJob. The top navigation bar includes 'Search (Ctrl+ /)', 'Add', 'Refresh', 'Logs', 'Delete', and 'Properties' buttons. The left sidebar under 'SETTINGS' has sections for Application settings, Authentication / Authorization (which is selected), Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' section is highlighted with a blue background. The main content area displays the 'WebJobs' title with a gear icon and a brief description: 'WebJobs provide an easy way to run scripts or programs as background processes'. A table lists three WebJobs:

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Ready	0 0/20 * * *
myTriggeredWeb...	Triggered	Ready	n/a
myContinuousW...	Continuous	Pending Restart	n/a

NCRONTAB expressions

You can enter a [NCRONTAB expression](#) in the portal or include a `settings.job` file at the root of your WebJob .zip file, as in the following example:

```
{  
  "schedule": "0 */15 * * *"  
}
```

To learn more, see [Scheduling a triggered WebJob](#).

NOTE

The default time zone used to run CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression run based on another time zone, create an app setting for your function app named WEBSITE_TIME_ZONE. To learn more, see [NCRONTAB time zones](#).

View the job history

1. Select the WebJob you want to see history for, and then select the **Logs** button.

WebApp0909 - WebJobs

App Service

Search (Ctrl+ /)

Add Refresh Logs Delete Run Properties

SETTINGS

- Application settings
- Authentication / Authorization
- Backups
- Custom domains
- SSL certificates
- Networking
- Scale up (App Service plan)

WebJobs

WebJobs provide an easy way to run scripts or programs as background processes in your app.

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Completed 5 min ago	0 0/20 * * *
myTriggeredWeb...	Triggered	Completed Just now	n/a
myContinuousW...	Continuous	Running	n/a

2. In the **WebJob Details** page, select a time to see details for one run.

Microsoft Azure WebJobs

WebJobs

WebJob Details myTriggeredWebJob

Run command: ConsoleApp1.exe

Recent job runs

TIMING	STATUS
13 minutes ago (203 ms running time)	Success
16 minutes ago (361 ms running time)	Success

Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables.

3. In the **WebJob Run Details** page, select **Toggle Output** to see the text of the log contents.

Microsoft Azure WebJobs

WebJobs / myTriggeredWebJob

WebJob Run Details myTriggeredWebJob

Success 20 minutes ago (203 ms running time)
Run ID: 201709112047171935

Toggle Output

download

```
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Status changed to Initializing  
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Run script 'ConsoleApp1.exe' with script host -  
'WindowsScriptHost'  
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Status changed to Running  
[09/11/2017 20:47:17 > 9ed5b3: INFO] Hello World!  
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Status changed to Success
```

Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables.

To see the output text in a separate browser window, select **download**. To download the text itself, right-click **download** and use your browser options to save the file contents.

4. Select the **WebJobs** breadcrumb link at the top of the page to go to a list of WebJobs.

Microsoft Azure WebJobs

WebJobs / myTriggeredWebJob

ites.net/azurejobs/#/jobs

WebJobs

NAME	STATUS	LAST RUN TIME
myScheduledWebJob	Success	14 minutes ago (328 ms)
myTriggeredWebJob	Success	27 minutes ago (203 ms)
myContinuousWebJob	Running	Runs continuously

Next steps

The Azure WebJobs SDK can be used with WebJobs to simplify many programming tasks. For more information,

see [What is the WebJobs SDK](#).

Develop and deploy WebJobs using Visual Studio - Azure App Service

12/2/2019 • 10 minutes to read • [Edit Online](#)

This article explains how to use Visual Studio to deploy a Console Application project to a web app in [App Service](#) as an [Azure WebJob](#). For information about how to deploy WebJobs by using the [Azure portal](#), see [Run Background tasks with WebJobs](#).

You can publish multiple WebJobs to a single web app. Make sure that each WebJob in a web app has a unique name.

Version 3.x of the [Azure WebJobs SDK](#) lets you develop WebJobs that run as either .NET Core apps or .NET Framework apps, while version 2.x supports only the .NET Framework. The way that you deploy a WebJobs project is different for .NET Core projects versus .NET Framework ones.

WebJobs as .NET Core console apps

When using version 3.x of the WebJobs, you can create and publish WebJobs as .NET Core console apps. For step-by-step instructions to create and publish a .NET Core console application to Azure as a WebJob, see [Get started with the Azure WebJobs SDK for event-driven background processing](#).

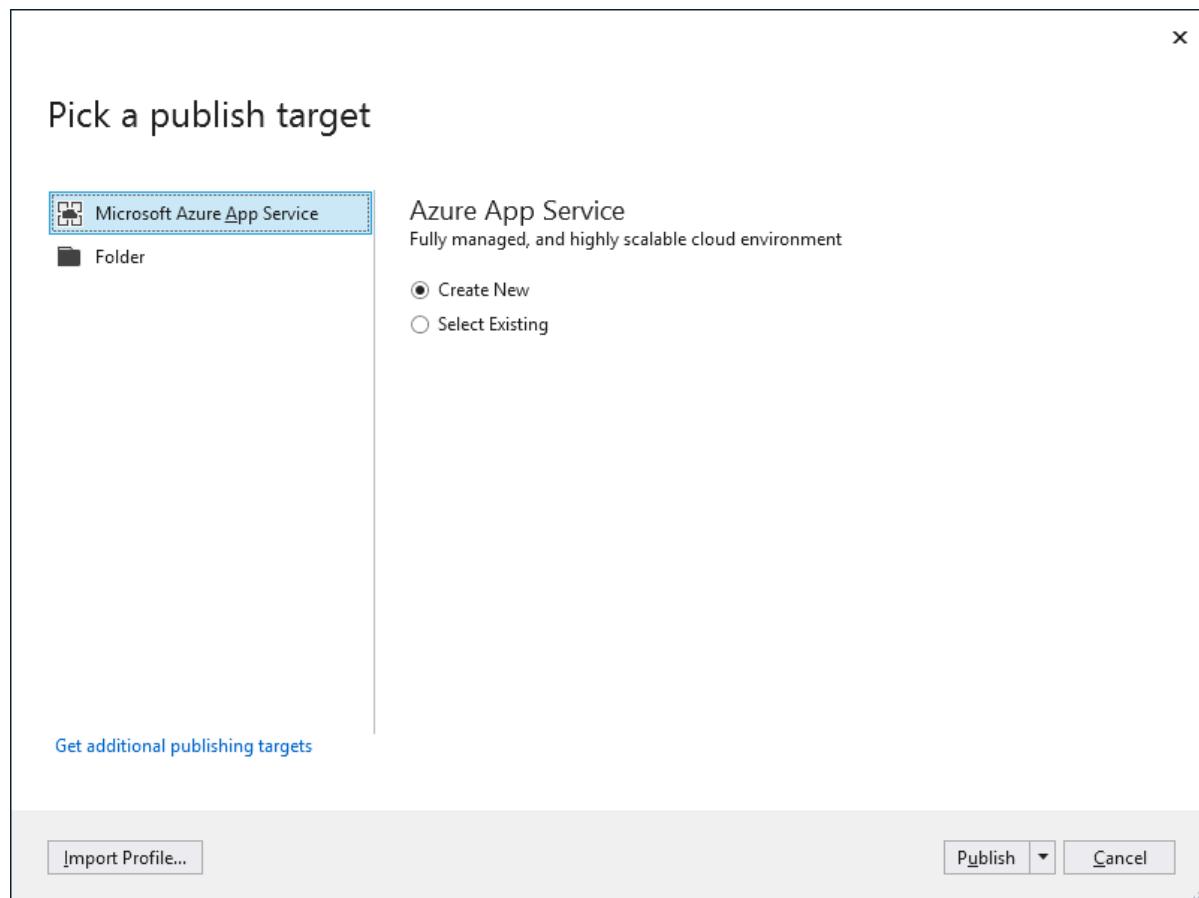
NOTE

.NET Core WebJobs cannot be linked with web projects. If you need to deploy your WebJob with a web app, you should [create your WebJob as a .NET Framework console app](#).

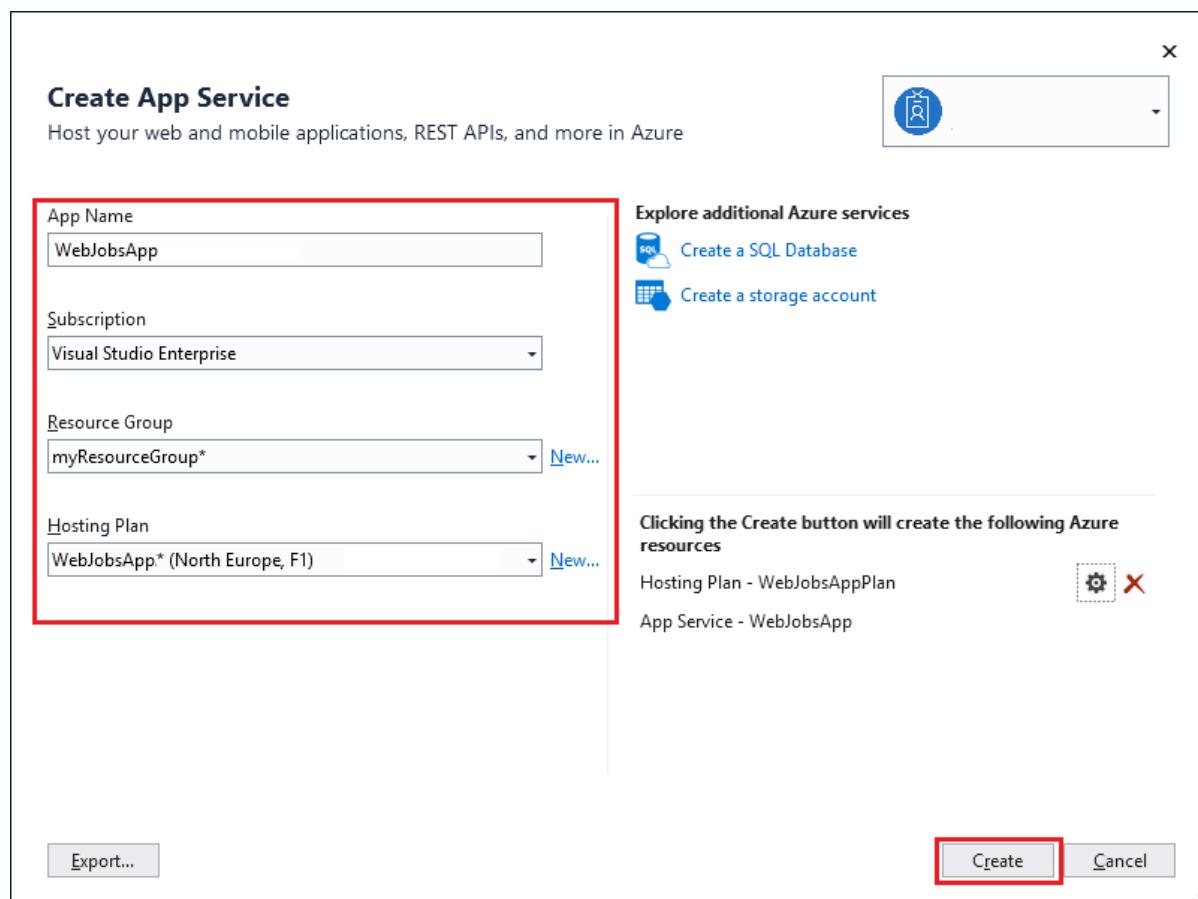
Deploy to Azure App Service

Publishing a .NET Core WebJob to App Service from Visual Studio uses the same tooling as publishing an ASP.NET Core app.

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog, select **Microsoft Azure App Service**, choose **Create New**, and then select **Publish**.



3. In the **Create App Service** dialog, use the hosting settings as specified in the table below the image:



SETTING	SUGGESTED VALUE	DESCRIPTION
---------	-----------------	-------------

SETTING	SUGGESTED VALUE	DESCRIPTION
App Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource Group	myResourceGroup	Name of the resource group in which to create your function app. Choose New to create a new resource group.
Hosting Plan	App Service plan	An App Service plan specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan. App Service plans define the region, instance size, scale count, and SKU (Free, Shared, Basic, Standard, or Premium). Choose New to create a new App Service plan.

- Click **Create** to create a WebJob and related resources in Azure with these settings and deploy your project code.

WebJob types

By default, a WebJob published from a .NET Core console project runs only when triggered or on demand. You can also update the project to [run on a schedule](#) or run continuously.

NOTE

A web app can time out after 20 minutes of inactivity. Only requests to the actual web app reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (

`https://<app_name>.scm.azurewebsites.net`) don't reset the timer. If your app runs continuous or scheduled (Timer trigger) WebJobs, enable **Always On** to ensure that the WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium [pricing tiers](#).

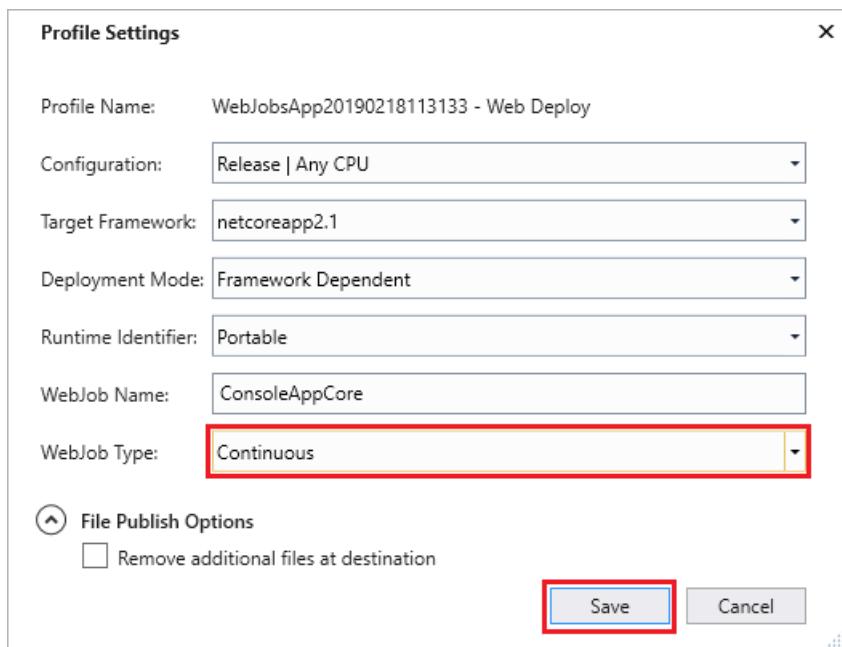
Scheduled execution

When you publish a .NET Core console application to Azure, a new `settings.job` file is added to the project. Use this file to set an execution schedule for your WebJob. For more information, see [Scheduling a triggered WebJob](#).

Continuous execution

You can use Visual Studio to change the WebJob to run continuously when Always On is enabled in Azure.

- If you haven't already done so, [publish the project to Azure](#).
- In **Solution Explorer**, right-click the project and select **Publish**.
- In the **Publish** tab, choose **Settings**.
- In the **Profile Settings** dialog, choose **Continuous** for **WebJob Type**, and choose **Save**.



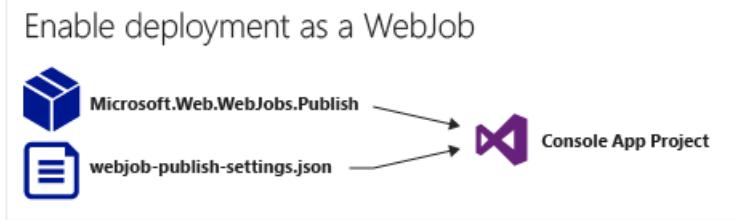
5. Select **Publish** to republish the WebJob with the updated settings.

WebJobs as .NET Framework console apps

When Visual Studio deploys a WebJobs-enabled .NET Framework Console Application project, it copies runtime files to the appropriate folder in the web app (*App_Data/jobs/continuous* for continuous WebJobs and *App_Data/jobs/triggered* for scheduled or on-demand WebJobs).

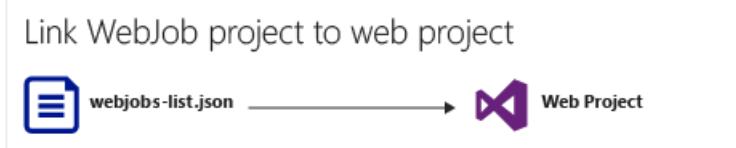
A WebJobs-enabled project has the following items added to it:

- The [Microsoft.Web.WebJobs.Publish](#) NuGet package.
- A [webjob-publish-settings.json](#) file that contains deployment and scheduler settings.



You can add these items to an existing Console Application project or use a template to create a new WebJobs-enabled Console Application project.

You can deploy a project as a WebJob by itself, or link it to a web project so that it automatically deploys whenever you deploy the web project. To link projects, Visual Studio includes the name of the WebJobs-enabled project in a [webjobs-list.json](#) file in the web project.



Prerequisites

If you're using Visual Studio 2015, install the [Azure SDK for .NET \(Visual Studio 2015\)](#).

If you're using Visual Studio 2017, install the [Azure development workload](#).

Enable WebJobs deployment for an existing Console Application project

You have two options:

- [Enable automatic deployment with a web project.](#)

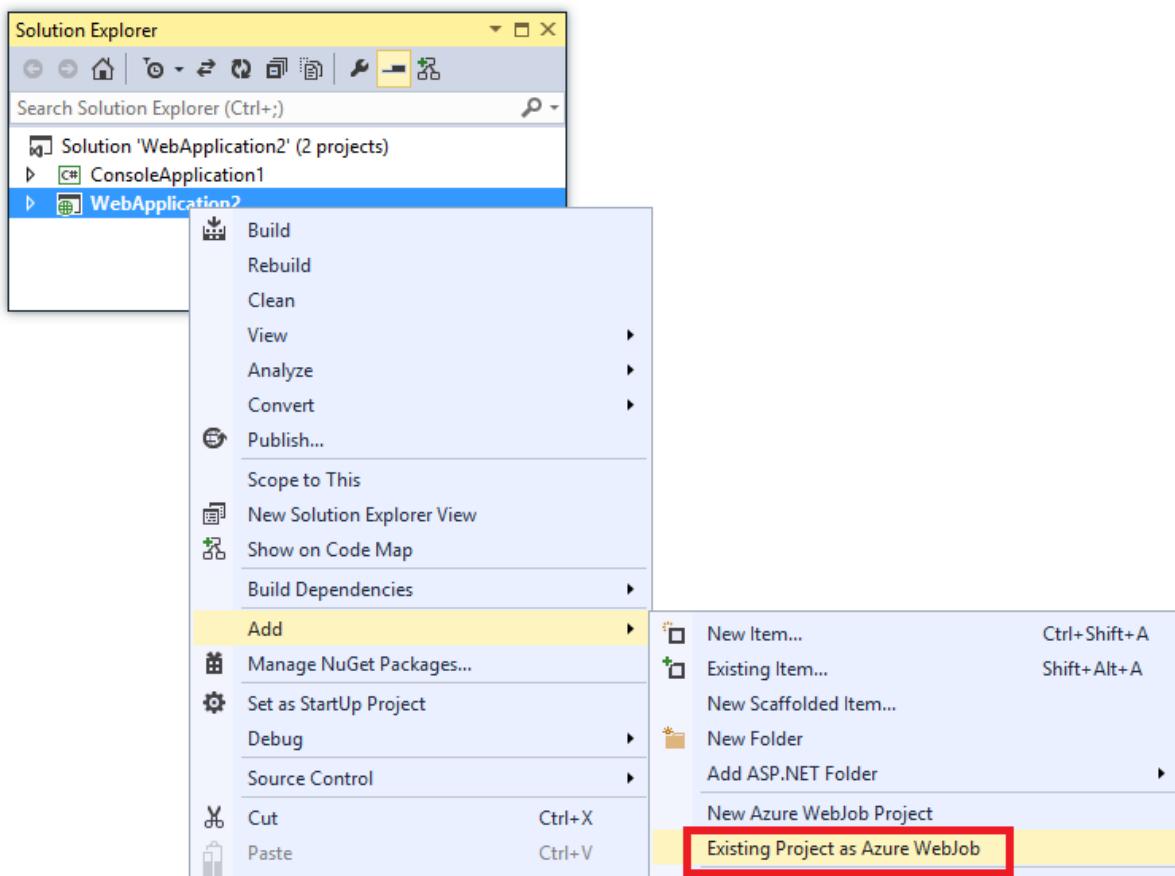
Configure an existing Console Application project so that it automatically deploys as a WebJob when you deploy a web project. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

- [Enable deployment without a web project.](#)

Configure an existing Console Application project to deploy as a WebJob by itself, with no link to a web project. Use this option when you want to run a WebJob in a web app by itself, with no web application running in the web app. You might want to do this in order to be able to scale your WebJob resources independently of your web application resources.

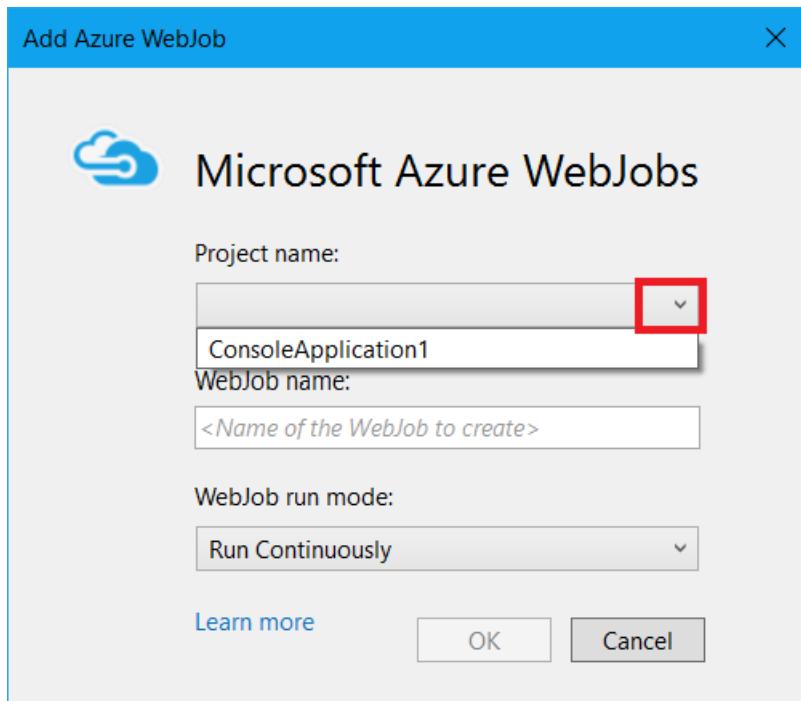
Enable automatic WebJobs deployment with a web project

1. Right-click the web project in **Solution Explorer**, and then click **Add > Existing Project as Azure WebJob.**



The Add Azure WebJob dialog box appears.

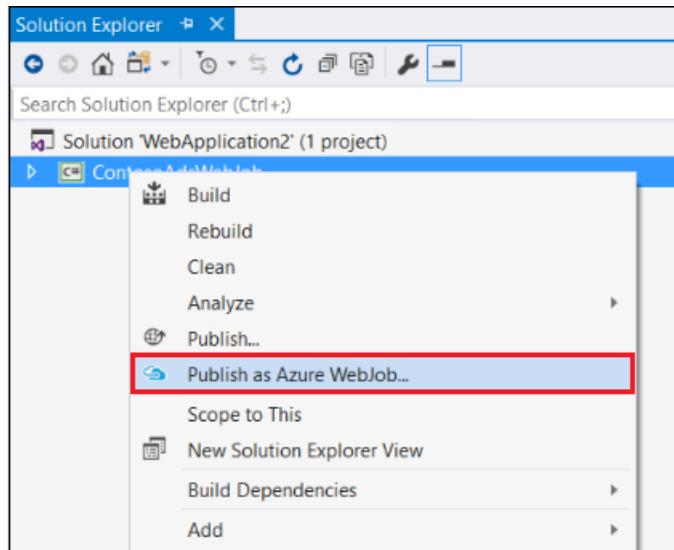
2. In the **Project name** drop-down list, select the Console Application project to add as a WebJob.



3. Complete the [Add Azure WebJob](#) dialog, and then click **OK**.

Enable WebJobs deployment without a web project

1. Right-click the Console Application project in **Solution Explorer**, and then click **Publish as Azure WebJob....**



The [Add Azure WebJob](#) dialog box appears, with the project selected in the **Project name** box.

2. Complete the [Add Azure WebJob](#) dialog box, and then click **OK**.

The **Publish Web** wizard appears. If you do not want to publish immediately, close the wizard. The settings that you've entered are saved for when you do want to [deploy the project](#).

Create a new WebJobs-enabled project

To create a new WebJobs-enabled project, you can use the Console Application project template and enable WebJobs deployment as explained in [the previous section](#). As an alternative, you can use the WebJobs new-project template:

- [Use the WebJobs new-project template for an independent WebJob](#)

Create a project and configure it to deploy by itself as a WebJob, with no link to a web project. Use this option when you want to run a WebJob in a web app by itself, with no web application running in the web

app. You might want to do this in order to be able to scale your WebJob resources independently of your web application resources.

- [Use the WebJobs new-project template for a WebJob linked to a web project](#)

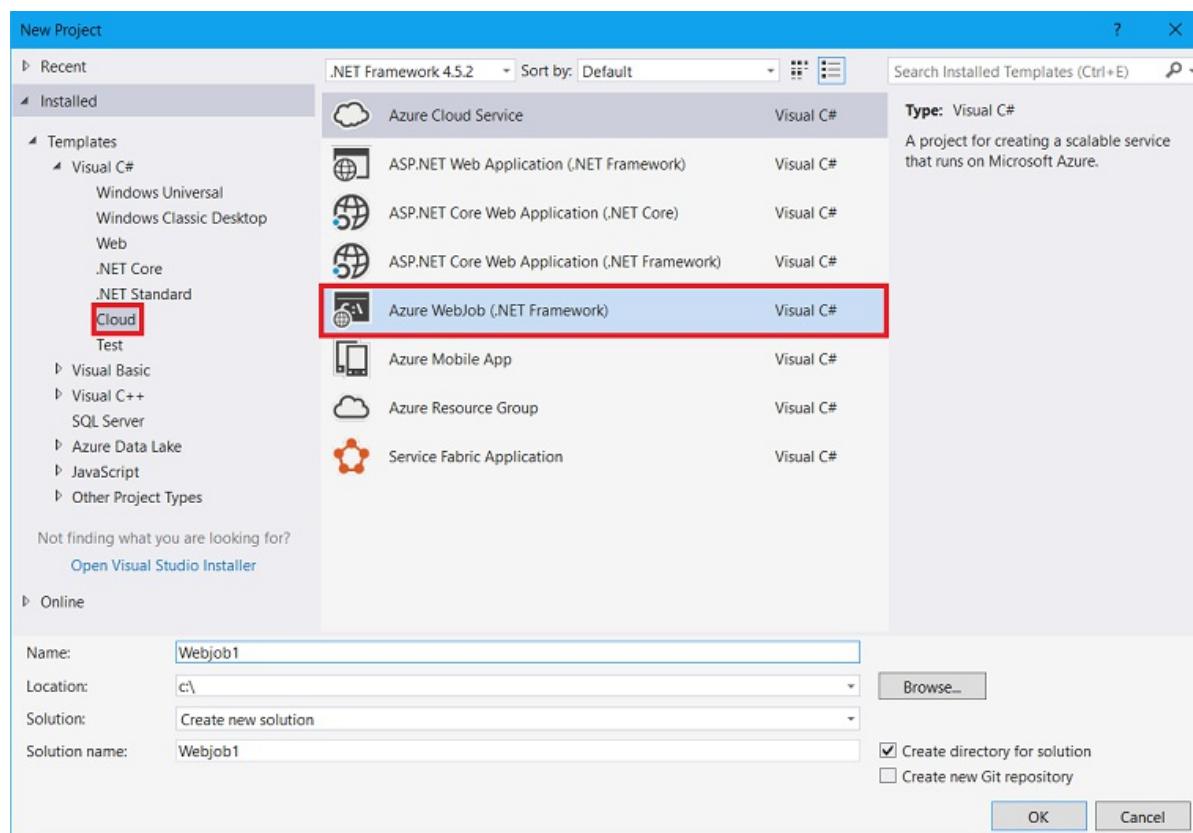
Create a project that is configured to deploy automatically as a WebJob when a web project in the same solution is deployed. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

NOTE

The WebJobs new-project template automatically installs NuGet packages and includes code in *Program.cs* for the [WebJobs SDK](#). If you don't want to use the WebJobs SDK, remove or change the `host.RunAndBlock` statement in *Program.cs*.

Use the WebJobs new-project template for an independent WebJob

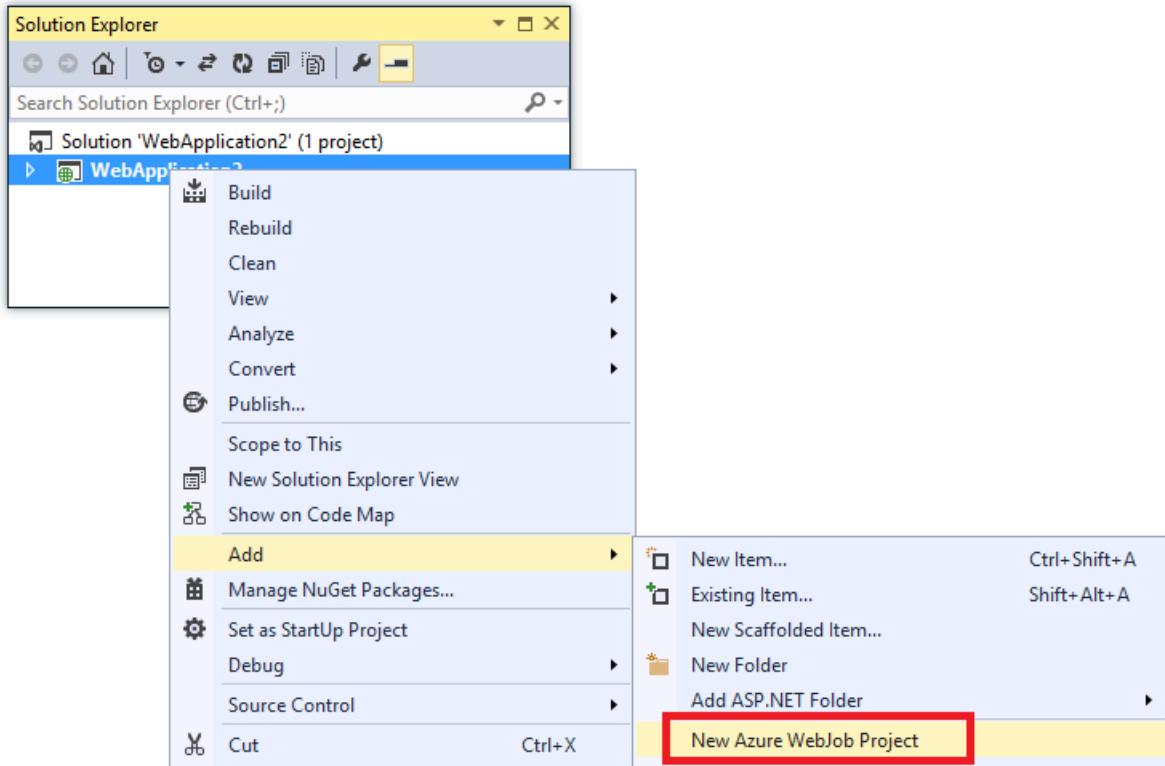
1. Click **File > New Project**, and then in the **New Project** dialog box click **Cloud > Azure WebJob (.NET Framework)**.



2. Follow the directions shown earlier to [make the Console Application project an independent WebJobs project](#).

Use the WebJobs new-project template for a WebJob linked to a web project

1. Right-click the web project in **Solution Explorer**, and then click **Add > New Azure WebJob Project**.

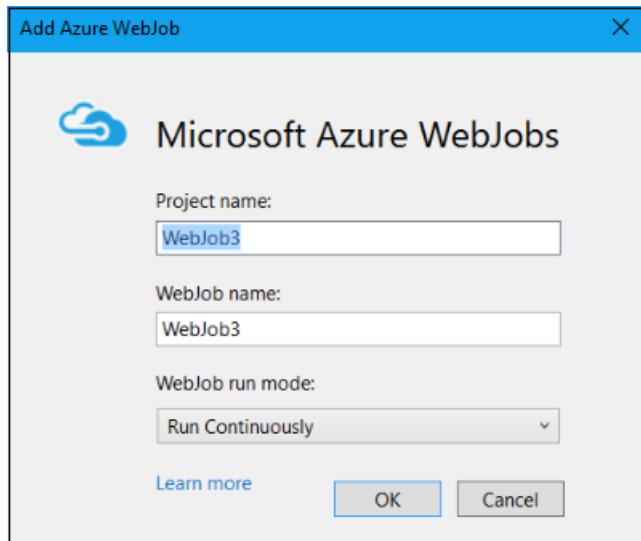


The [Add Azure WebJob](#) dialog box appears.

2. Complete the [Add Azure WebJob](#) dialog box, and then click **OK**.

The Add Azure WebJob dialog

The [Add Azure WebJob](#) dialog lets you enter the WebJob name and run mode setting for your WebJob.



The fields in this dialog correspond to fields on the [Add WebJob](#) dialog of the Azure portal. For more information, see [Run Background tasks with WebJobs](#).

NOTE

- For information about command-line deployment, see [Enabling Command-line or Continuous Delivery of Azure WebJobs](#).
- If you deploy a WebJob and then decide you want to change the type of WebJob and redeploy, you'll need to delete the `webjobs-publish-settings.json` file. This will make Visual Studio show the publishing options again, so you can change the type of WebJob.
- If you deploy a WebJob and later change the run mode from continuous to non-continuous or vice versa, Visual Studio creates a new WebJob in Azure when you redeploy. If you change other scheduling settings but leave run mode the same or switch between Scheduled and On Demand, Visual Studio updates the existing job rather than create a new one.

webjob-publish-settings.json

When you configure a Console Application for WebJobs deployment, Visual Studio installs the [Microsoft.Web.WebJobs.Publish](#) NuGet package and stores scheduling information in a `webjob-publish-settings.json` file in the project *Properties* folder of the WebJobs project. Here is an example of that file:

```
{  
  "$schema": "http://schemastore.org/schemas/json/webjob-publish-settings.json",  
  "webJobName": "WebJob1",  
  "startTime": "null",  
  "endTime": "null",  
  "jobRecurrenceFrequency": "null",  
  "interval": null,  
  "runMode": "Continuous"  
}
```

You can edit this file directly, and Visual Studio provides IntelliSense. The file schema is stored at <https://schemastore.org> and can be viewed there.

webjobs-list.json

When you link a WebJobs-enabled project to a web project, Visual Studio stores the name of the WebJobs project in a `webjobs-list.json` file in the web project's *Properties* folder. The list might contain multiple WebJobs projects, as shown in the following example:

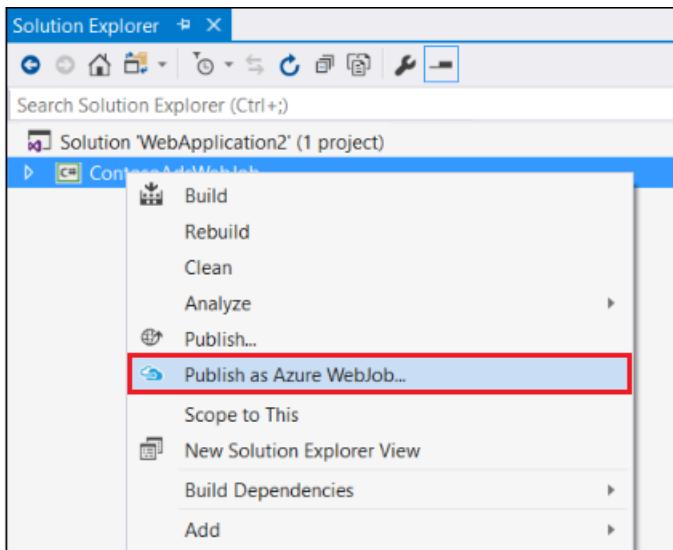
```
{  
  "$schema": "http://schemastore.org/schemas/json/webjobs-list.json",  
  "WebJobs": [  
    {  
      "filePath": "../ConsoleApplication1/ConsoleApplication1.csproj"  
    },  
    {  
      "filePath": "../WebJob1/WebJob1.csproj"  
    }  
  ]  
}
```

You can edit this file directly, and Visual Studio provides IntelliSense. The file schema is stored at <https://schemastore.org> and can be viewed there.

Deploy a WebJobs project

A WebJobs project that you have linked to a web project deploys automatically with the web project. For information about web project deployment, see [How-to guides > Deploy app](#) in the left navigation.

To deploy a WebJobs project by itself, right-click the project in **Solution Explorer** and click **Publish as Azure WebJob....**



For an independent WebJob, the same **Publish Web** wizard that is used for web projects appears, but with fewer settings available to change.

Scheduling a triggered WebJob

WebJobs uses a *settings.job* file to determine when a WebJob is run. Use this file to set an execution schedule for your WebJob. The following example runs every hour from 9 AM to 5 PM:

```
{  
    "schedule": "0 0 9-17 * * *"  
}
```

This file must be located at the root of the WebJobs folder, along side your WebJob's script, such as

`wwwroot\app_data\jobs\triggered\{job name}` or `wwwroot\app_data\jobs\continuous\{job name}`. When you deploy a WebJob from Visual Studio, mark your `settings.job` file properties as **Copy if newer**.

When you [create a WebJob from the Azure portal](#), the `settings.job` file is created for you.

NOTE

A web app can time out after 20 minutes of inactivity. Only requests to the actual web app reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (

https://<app_name>.scm.azurewebsites.net

CRON expressions

WebJobs uses the same CRON expressions for scheduling as the timer trigger in Azure Functions. To learn more about CRON support, see the [timer trigger reference article](#).

NOTE

The default time zone used to run CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression run based on another time zone, create an app setting for your function app named WEBSITE_TIME_ZONE. To learn more, see [NCRONTAB time zones](#).

setting.job reference

The following settings are supported by WebJobs:

SETTING	TYPE	DESCRIPTION
<code>is_in_place</code>	All	Allows the job to run in place without being first copied to a temp folder. To learn more, see WebJobs working directory .
<code>is_singleton</code>	Continuous	Only run the WebJobs on a single instance when scaled out. To learn more, see Set a continuous job as singleton .
<code>schedule</code>	Triggered	Run the WebJob on a CRON-based schedule. To learn more, see the timer trigger reference article .
<code>stopping_wait_time</code>	All	Allows control of the shutdown behavior. To learn more, see Graceful shutdown .

Next steps

[Learn more about the WebJobs SDK](#)

Get started with the Azure WebJobs SDK for event-driven background processing

2/25/2020 • 14 minutes to read • [Edit Online](#)

This article shows how to use Visual Studio 2019 to create an Azure WebJobs SDK project, run it locally, and then deploy it to [Azure App Service](#). Version 3.x of the WebJobs SDK supports both .NET Core and .NET Framework console apps. To learn more about working with the WebJobs SDK, see [How to use the Azure WebJobs SDK for event-driven background processing](#).

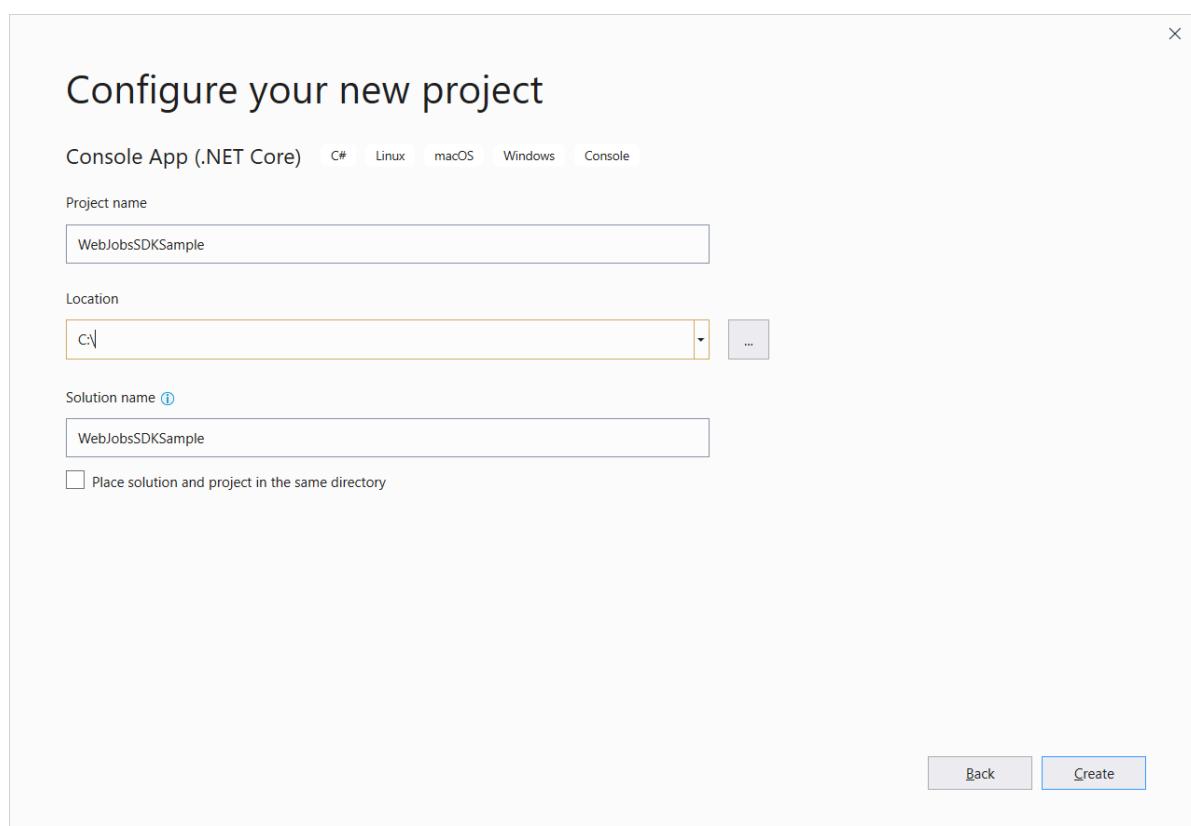
This article shows you how to deploy WebJobs as a .NET Core console app. To deploy WebJobs as a .NET Framework console app, see [WebJobs as .NET Framework console apps](#). If you are interested in WebJobs SDK version 2.x, which only supports .NET Framework, see [Develop and deploy WebJobs using Visual Studio - Azure App Service](#).

Prerequisites

- [Install Visual Studio 2019](#) with the **Azure development** workload. If you already have Visual Studio but don't have that workload, add the workload by selecting **Tools > Get Tools and Features**.
- You must have [an Azure account](#) to publish your WebJobs SDK project to Azure.

Create a project

1. In Visual Studio, select **Create a New Project**.
2. Select **Console App (.NET Core)**.
3. Name the project *WebJobsSDKSample*, and then select **Create**.



WebJobs NuGet packages

1. Install the latest stable 3.x version of the `Microsoft.Azure.WebJobs.Extensions` NuGet package, which includes `Microsoft.Azure.WebJobs`.

Here's the **Package Manager Console** command for version 3.0.2:

```
Install-Package Microsoft.Azure.WebJobs.Extensions -version 3.0.2
```

Create the Host

The host is the runtime container for functions that listens for triggers and calls functions. The following steps create a host that implements `IHost`, which is the Generic Host in ASP.NET Core.

1. In `Program.cs`, add a `using` statement:

```
using Microsoft.Extensions.Hosting;
```

2. Replace the `Main` method with the following code:

```
static void Main(string[] args)
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    var host = builder.Build();
    using (host)
    {
        host.Run();
    }
}
```

In ASP.NET Core, host configurations are set by calling methods on the `HostBuilder` instance. For more information, see [.NET Generic Host](#). The `ConfigureWebJobs` extension method initializes the WebJobs host. In `ConfigureWebJobs`, you initialize specific WebJobs extensions and set properties of those extensions.

Enable console logging

In this section, you set up console logging that uses the [ASP.NET Core logging framework](#).

1. Install the latest stable version of the `Microsoft.Extensions.Logging.Console` NuGet package, which includes `Microsoft.Extensions.Logging`.

Here's the **Package Manager Console** command for version 2.2.0:

```
Install-Package Microsoft.Extensions.Logging.Console -version 2.2.0
```

2. In `Program.cs`, add a `using` statement:

```
using Microsoft.Extensions.Logging;
```

3. Call the `ConfigureLogging` method on `HostBuilder`. The `AddConsole` method adds console logging to the

configuration.

```
builder.ConfigureLogging((context, b) =>
{
    b.AddConsole();
});
```

The `Main` method now looks like this:

```
static void Main(string[] args)
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureLogging((context, b) =>
    {
        b.AddConsole();
    });
    var host = builder.Build();
    using (host)
    {
        host.Run();
    }
}
```

This update does the following:

- Disables [dashboard logging](#). The dashboard is a legacy monitoring tool, and dashboard logging is not recommended for high-throughput production scenarios.
- Adds the console provider with default [filtering](#).

Now, you can add a function that is triggered by messages arriving in an [Azure Storage queue](#).

Install the Storage binding extension

Starting with version 3.x, you must explicitly install the Storage binding extension required by the WebJobs SDK. In prior versions, the Storage bindings were included in the SDK.

1. Install the latest stable version of the [Microsoft.Azure.WebJobs.Extensions.Storage](#) NuGet package, version 3.x.

Here's the **Package Manager Console** command for version 3.0.4:

```
Install-Package Microsoft.Azure.WebJobs.Extensions.Storage -Version 3.0.4
```

2. In the `ConfigureWebJobs` extension method, call the `AddAzureStorage` method on the `HostBuilder` instance to initialize the Storage extension. At this point, the `ConfigureWebJobs` method looks like the following example:

```
builder.ConfigureWebJobs(b =>
{
    b.AddAzureStorageCoreServices();
    b.AddAzureStorage();
});
```

Create a function

1. Right-click the project, select **Add > New Item...**, choose **Class**, name the new C# class file *Functions.cs*, and select **Add**.
2. In *Functions.cs*, replace the generated template with the following code:

```
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;

namespace WebJobsSDKSample
{
    public class Functions
    {
        public static void ProcessQueueMessage([QueueTrigger("queue")] string message, ILogger logger)
        {
            logger.LogInformation(message);
        }
    }
}
```

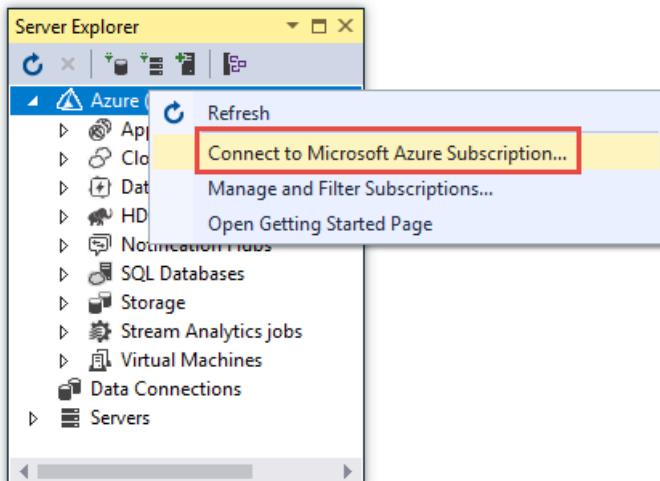
The `QueueTrigger` attribute tells the runtime to call this function when a new message is written on an Azure Storage queue called `queue`. The contents of the queue message are provided to the method code in the `message` parameter. The body of the method is where you process the trigger data. In this example, the code just logs the message.

The `message` parameter doesn't have to be a string. You can also bind to a JSON object, a byte array, or a `CloudQueueMessage` object. [See Queue trigger usage](#). Each binding type (such as queues, blobs, or tables) has a different set of parameter types that you can bind to.

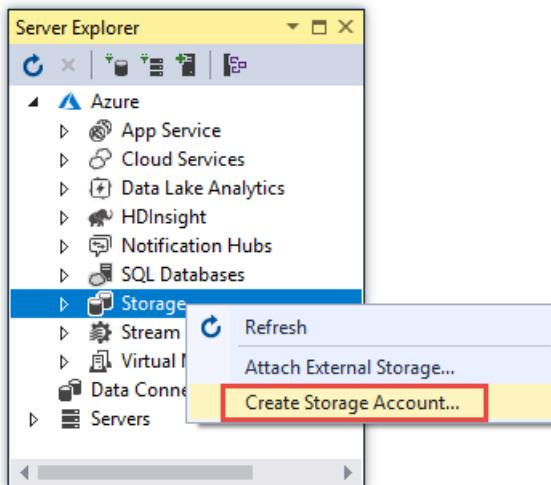
Create a storage account

The Azure Storage emulator that runs locally doesn't have all of the features that the WebJobs SDK needs. So in this section you create a storage account in Azure and configure the project to use it. If you already have a storage account, skip down to step 6.

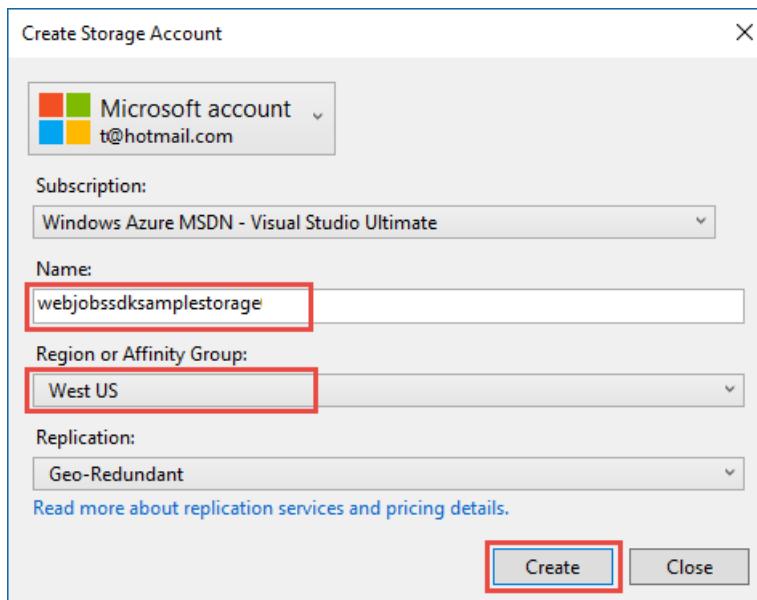
1. Open **Server Explorer** in Visual studio and sign in to Azure. Right-click the **Azure** node, and then select **Connect to Microsoft Azure Subscription**.



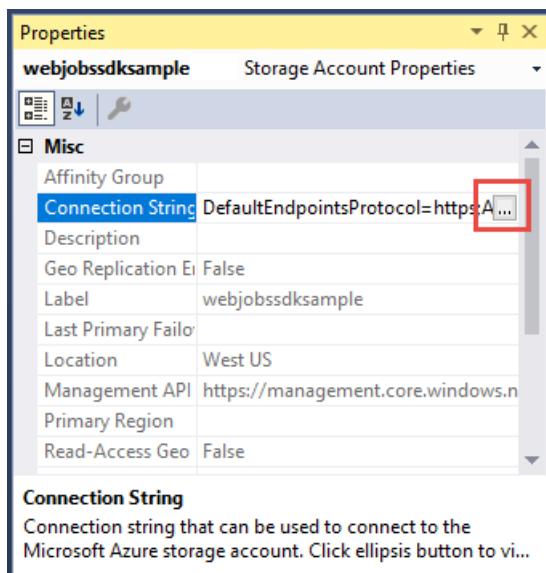
2. Under the **Azure** node in **Server Explorer**, right-click **Storage**, and then select **Create Storage account**.



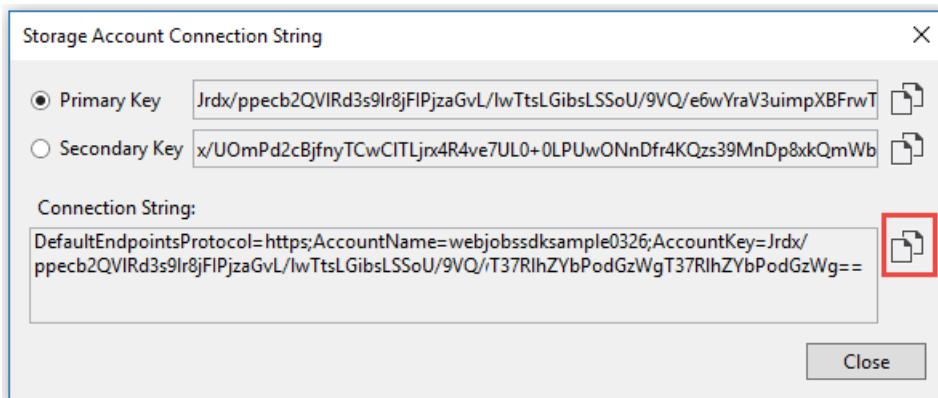
3. In the **Create Storage Account** dialog box, enter a unique name for the storage account.
4. Choose the same **Region** that you created your App Service app in, or a region close to you.
5. Select **Create**.



6. Under the **Storage** node in **Server Explorer**, select the new Storage account. In the **Properties** window, select the ellipsis (...) at the right of the **Connection String** value field.



7. Copy the connection string, and save this value somewhere that you can copy it again readily.



Configure storage to run locally

The WebJobs SDK looks for the storage connection string in the Application Settings in Azure. When you run locally, it looks for this value in the local configuration file or in environment variables.

1. Right-click the project, select **Add > New Item...**, choose **JavaScript JSON configuration file**, name the new file *appsettings.json* file, and select **Add**.
2. In the new file, add a `AzureWebJobsStorage` field, as in the following example:

```
{  
    "AzureWebJobsStorage": "{storage connection string}"  
}
```

3. Replace `{storage connection string}` with the connection string that you copied earlier.
4. Select the *appsettings.json* file in Solution Explorer and in the **Properties** window, set **Copy to Output Directory** to **Copy if newer**.

Later, you'll add the same connection string app setting in your app in Azure App Service.

Test locally

In this section, you build and run the project locally and trigger the function by creating a queue message.

1. Press **Ctrl+F5** to run the project.

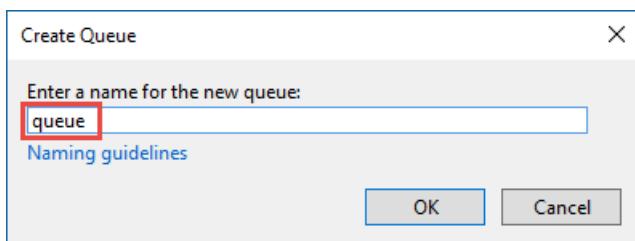
The console shows that the runtime found your function and is waiting for queue messages to trigger it. The following output is generated by the v3.x host:

```
info: Microsoft.Azure.WebJobs.Hosting.JobHostService[0]  
      Starting JobHost  
info: Host.Startup[0]  
      Found the following functions:  
      WebJobsSDKSample.Functions.ProcessQueueMessage  
  
info: Host.Startup[0]  
      Job host started  
Application started. Press Ctrl+C to shut down.  
Hosting environment: Development  
Content root path: C:\WebJobsSDKSample\WebJobsSDKSample\bin\Debug\netcoreapp2.1\
```

2. Close the console window.
3. In **Server Explorer** in Visual Studio, expand the node for your new storage account, and then right-click

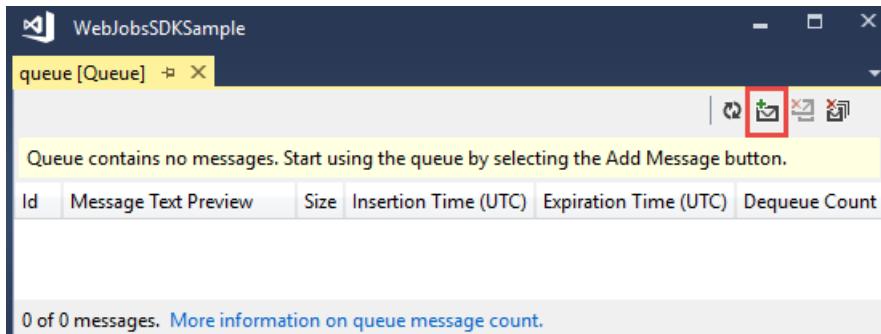
Queues.

4. Select **Create Queue**.
5. Enter *queue* as the name for the queue, and then select **OK**.

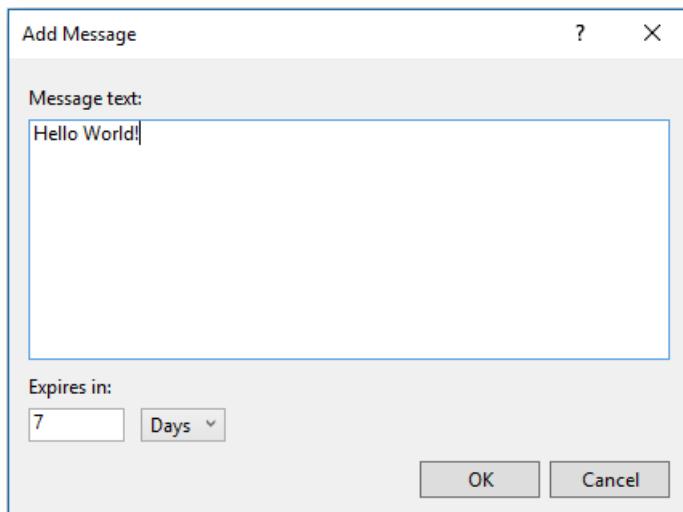


6. Right-click the node for the new queue, and then select **View Queue**.

7. Select the **Add Message** icon.



8. In the **Add Message** dialog, enter *Hello World!* as the **Message text**, and then select **OK**. There is now a message in the queue.



9. Run the project again.

Because you used the `queueTrigger` attribute in the `ProcessQueueMessage` function, the WeJobs SDK runtime listens for queue messages when it starts up. It finds a new queue message in the queue named *queue* and calls the function.

Due to [queue polling exponential backoff](#), it might take as long as 2 minutes for the runtime to find the message and invoke the function. This wait time can be reduced by running in [development mode](#).

The console output looks like this:

```

info: Function.ProcessQueueMessage[0]
    Executing 'Functions.ProcessQueueMessage' (Reason='New queue message detected on 'queue'.', 
Id=2c319369-d381-43f3-aedf-ff538a4209b8)
info: Function.ProcessQueueMessage[0]
    Trigger Details: MessageId: b00a86dc-298d-4cd2-811f-98ec39545539, DequeueCount: 1,
InsertionTime: 1/18/2019 3:28:51 AM +00:00
info: Function.ProcessQueueMessage.User[0]
    Hello World!
info: Function.ProcessQueueMessage[0]
Executed 'Functions.ProcessQueueMessage' (Succeeded, Id=2c319369-d381-43f3-aedf-ff538a4209b8)

```

10. Close the console window.
11. Go back to the Queue window and refresh it. The message is gone, since it has been processed by your function running locally.

Add Application Insights logging

When the project runs in Azure, you can't monitor function execution by viewing console output. The monitoring solution we recommend is [Application Insights](#). For more information, see [Monitor Azure Functions](#).

In this section, you do the following tasks to set up Application Insights logging before you deploy to Azure:

- Make sure you have an App Service app and an Application Insights instance to work with.
- Configure the App Service app to use the Application Insights instance and the storage account that you created earlier.
- Set up the project for logging to Application Insights.

Create App Service app and Application Insights instance

1. If you don't already have an App Service app that you can use, [create one](#). When you create your app, you can also create a connected Application Insights resource. When you do this, the **APPINSIGHTS_INSTRUMENTATIONKEY** is set for you in your app.
2. If you don't already have an Application Insights resource that you can use, [create one](#). Set **Application type** to **General**, and skip the sections that follow [Copy the instrumentation key](#).
3. If you already have an Application Insights resource that you want to use, [copy the instrumentation key](#).

Configure app settings

1. In **Server Explorer** in Visual Studio, expand the **App Service** node under **Azure**.
2. Expand the resource group that your App Service app is in, and then right-click your App Service app.
3. Select **View Settings**.
4. In the **Connection Strings** box, add the following entry.

NAME	CONNECTION STRING	DATABASE TYPE
AzureWebJobsStorage	{the Storage connection string that you copied earlier}	Custom

5. If the **Application Settings** box doesn't have an Application Insights instrumentation key, add the one that you copied earlier. (The instrumentation key may already be there, depending on how you created the App Service app.)

NAME	VALUE
APPINSIGHTS_INSTRUMENTATIONKEY	{instrumentation key}

6. Replace *{instrumentation key}* with the instrumentation key from the Application Insights resource that you're using.
7. Select **Save**.
8. Add the Application Insights connection to the project so that you can run it locally. In the *appsettings.json* file, add an `APPINSIGHTS_INSTRUMENTATIONKEY` field, as in the following example:

```
{
  "AzureWebJobsStorage": "{storage connection string}",
  "APPINSIGHTS_INSTRUMENTATIONKEY": "{instrumentation key}"
}
```

Replace *{instrumentation key}* with the instrumentation key from the Application Insights resource that you're using.

9. Save your changes.

Add Application Insights logging provider

To take advantage of [Application Insights](#) logging, update your logging code to do the following:

- Add an Application Insights logging provider with default [filtering](#); all Information and higher-level logs goes to both the console and Application Insights when you're running locally.
- Put the [LoggerFactory](#) object in a `using` block to ensure that log output is flushed when the host exits.

1. Install the latest stable 3.x version of the NuGet package for the Application Insights logging provider:

```
Microsoft.Azure.WebJobs.Logging.ApplicationInsights
```

Here's the **Package Manager Console** command for version 3.0.2:

```
Install-Package Microsoft.Azure.WebJobs.Logging.ApplicationInsights -Version 3.0.2
```

2. Open *Program.cs* and replace the code in the `Main` method with the following code:

```

static void Main(string[] args)
{
    var builder = new HostBuilder();
    builder.UseEnvironment(EnvironmentName.Development);
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage();
    });
    builder.ConfigureLogging((context, b) =>
    {
        b.AddConsole();

        // If the key exists in settings, use it to enable Application Insights.
        string instrumentationKey = context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
        if (!string.IsNullOrEmpty(instrumentationKey))
        {
            b.AddApplicationInsightsWebJobs(o => o.InstrumentationKey = instrumentationKey);
        }
    });
    var host = builder.Build();
    using (host)
    {
        host.Run();
    }
}

```

This adds the Application Insights provider to the logging, using the key you added earlier to your app settings.

Test Application Insights logging

In this section, you run locally again to verify that logging data is now going to Application Insights as well as to the console.

1. Use **Server Explorer** in Visual Studio to create a queue message like you did [earlier](#), except enter *Hello App Insights!* as the message text.
 2. Run the project.
- The WebJobs SDK processes the queue message, and you see the logs in the console window.
3. Close the console window.
 4. Go to the [Azure portal](#) to view your Application Insights resource. Search for and select **Application Insights**.
 5. Choose your Application Insights instance.
 6. Select **Search**.

The screenshot shows the Azure Application Insights interface for a resource group named 'webapp'. The top navigation bar includes 'Home', 'Resource groups', 'webapp', and 'webapp'. The main content area is titled 'Application Insights - Last 24 hours (30 minute granularity) - ASP.NET web application'. A red box highlights the 'Search' button in the top navigation bar. Below it, a message says 'NEW - Learn whether users come back to your app with the Retention tool.' A section titled 'Essentials' displays various details about the application, such as Resource group (webapp), Type (ASP.NET), Location (West US 2), Subscription name (Windows Azure MSDN - Visual Studio Ultima...), and Subscription ID ({subscription ID}).

7. If you don't see the *Hello App Insights!* message, select **Refresh** periodically for several minutes. (Logs don't appear immediately, because it takes a while for the Application Insights client to flush the logs it processes.)

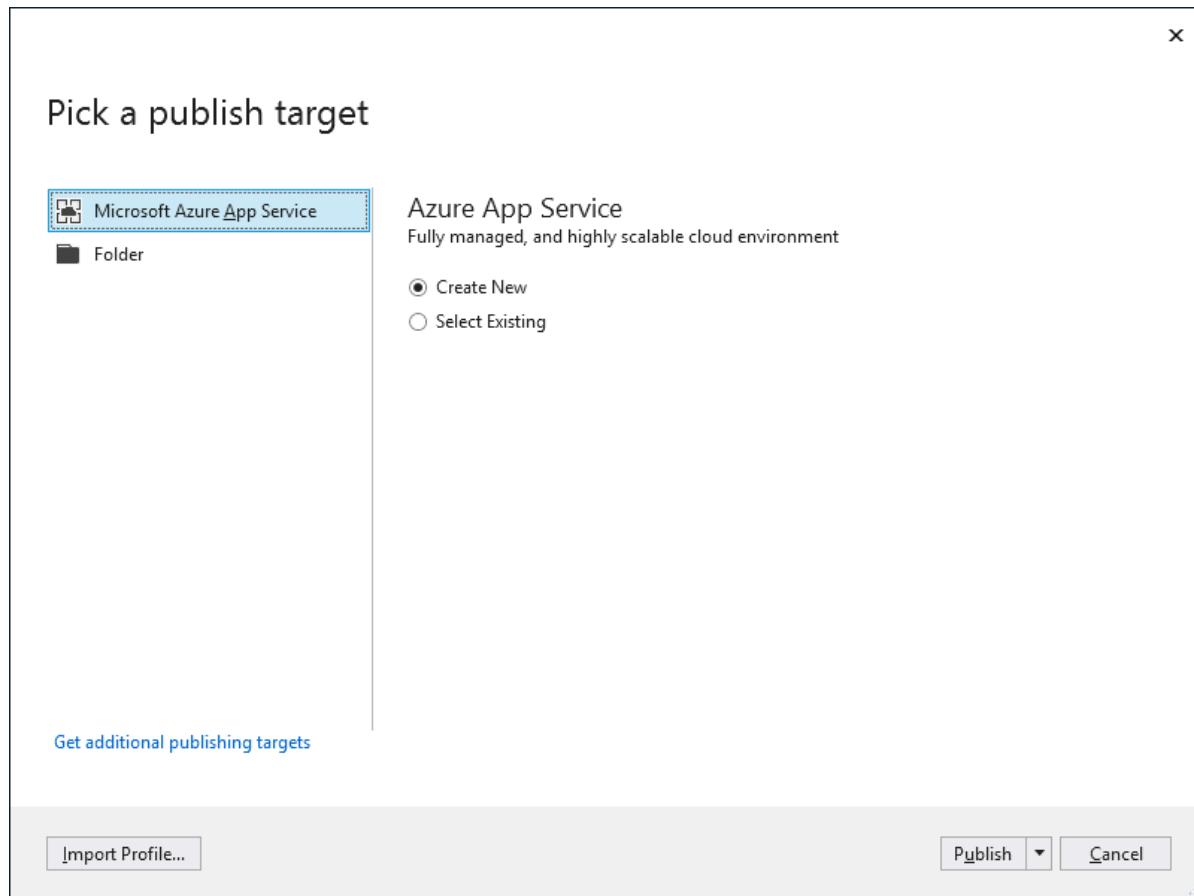
The screenshot shows the Azure Application Insights Metrics Explorer for the 'webapp' resource group. The top navigation bar includes 'Home', 'webapp', and 'Search'. The main content area is titled 'Last 24 hours (30 minute granularity) - webapp'. A red box highlights the 'Refresh' button in the top navigation bar. Below it, there's a search bar and filter options ('Filtered on Trace x, Request x, Page View x, Custom Event x, Exception x, Dependency x, Availability x'). The results summary shows '26 total results between 3/26/2018 2:07 PM and 3/27/2018 2:07 PM'. A chart shows event counts over time: 15 at 6 PM, 5 at 6 AM, and 2 at 12 PM. Below the chart, a legend indicates event types: TRACE (21), REQUEST (5), EXCEPTION (0), VIEW (0), EVENT (0), DEPENDENCY (0), and AVAIL (0). The 'Results' tab is selected, showing grouped results (3). One result is highlighted with a red box: '3/27/2018 2:03:14 PM - TRACE' followed by the message 'Hello Application Insights!' and 'Device type: PC Severity level: Informational'.

8. Close the console window.

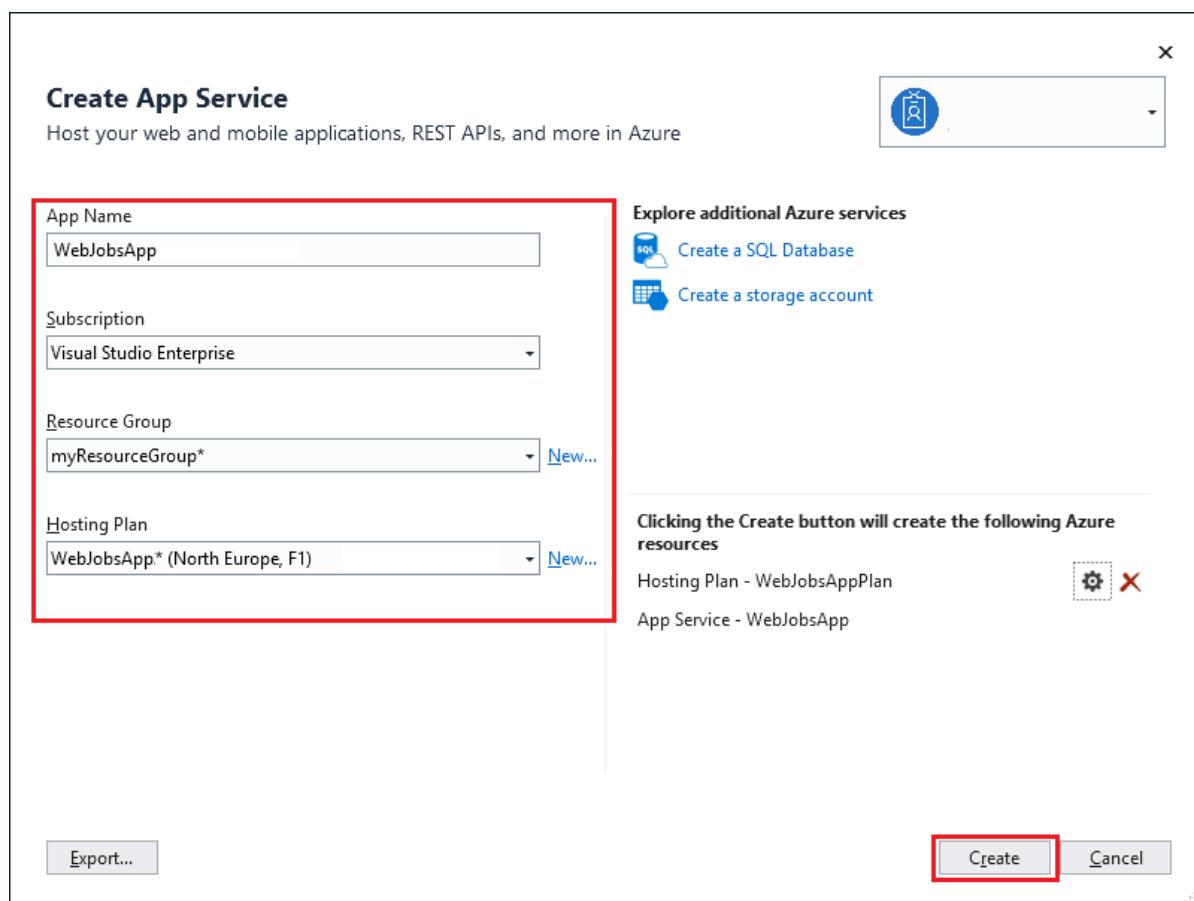
Deploy to Azure

During deployment, you create an app service instance in which to run your functions. When you publish a .NET Core console app to App Service in Azure, it automatically gets run as a WebJob. To learn more about publishing, see [Develop and deploy WebJobs using Visual Studio](#).

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog, select **Microsoft Azure App Service**, choose **Create New**, and then select **Publish**.



3. In the **Create App Service** dialog, use the hosting settings as specified in the table below the image:



SETTING	SUGGESTED VALUE	DESCRIPTION
App Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource Group	myResourceGroup	Name of the resource group in which to create your function app. Choose New to create a new resource group.
Hosting Plan	App Service plan	An App Service plan specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan. App Service plans define the region, instance size, scale count, and SKU (Free, Shared, Basic, Standard, or Premium). Choose New to create a new App Service plan.

- Click **Create** to create a WebJob and related resources in Azure with these settings and deploy your project code.

Trigger the function in Azure

- Make sure you're not running locally (close the console window if it's still open). Otherwise the local instance might be the first to process any queue messages you create.
- In the **Queue** page in Visual Studio, add a message to the queue as before.
- Refresh the **Queue** page, and the new message disappears because it has been processed by the function running in Azure.

TIP

When you're testing in Azure, use [development mode](#) to ensure that a queue trigger function is invoked right away and avoid delays due to [queue polling exponential backoff](#).

View logs in Application Insights

- Open the [Azure portal](#), and go to your Application Insights resource.
- Select **Search**.
- If you don't see the *Hello Azure!* message, select **Refresh** periodically for several minutes.

You see the logs from the function running in a WebJob, including the *Hello Azure!* text that you entered in the preceding section.

Add an input binding

Input bindings simplify code that reads data. For this example, the queue message will be a blob name and you'll use the blob name to find and read a blob in Azure Storage.

- In `Functions.cs`, replace the `ProcessQueueMessage` method with the following code:

```

public static void ProcessQueueMessage(
    [QueueTrigger("queue")] string message,
    [Blob("container/{queueTrigger}", FileAccess.Read)] Stream myBlob,
    ILogger logger)
{
    logger.LogInformation($"Blob name:{message} \n Size: {myBlob.Length} bytes");
}

```

In this code, `queueTrigger` is a [binding expression](#), which means it resolves to a different value at runtime.

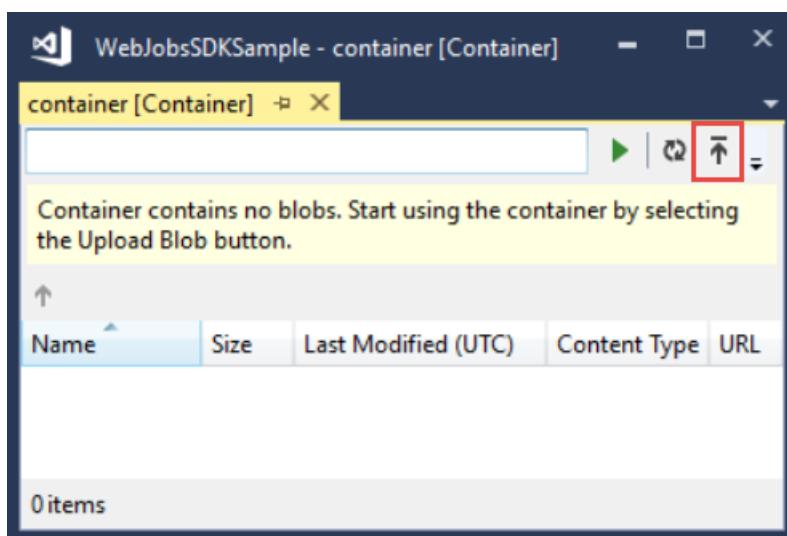
At runtime, it has the contents of the queue message.

2. Add a `using`:

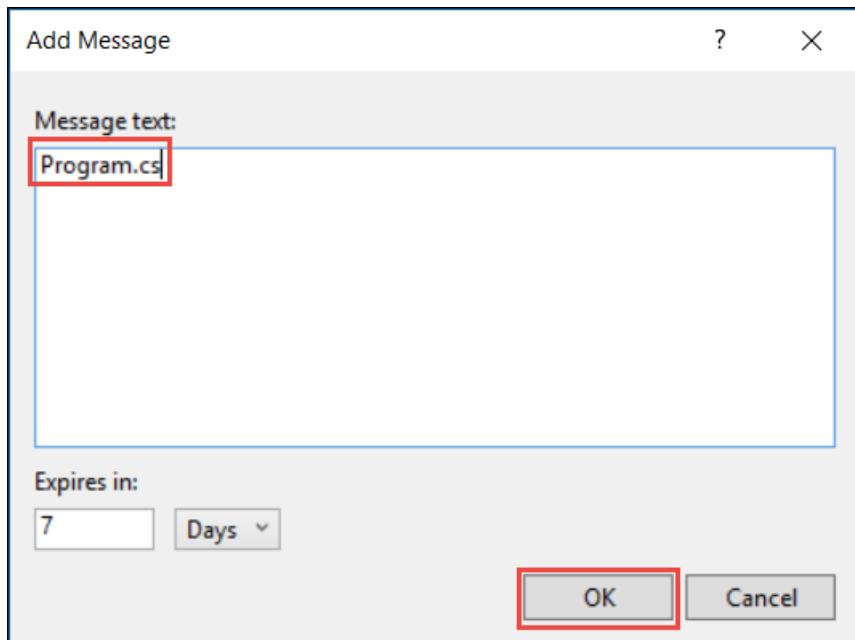
```
using System.IO;
```

3. Create a blob container in your storage account.

- In **Server Explorer** in Visual Studio, expand the node for your storage account, right-click **Blobs**, and then select **Create Blob Container**.
- In the **Create Blob Container** dialog, enter *container* as the container name, and then click **OK**.
- Upload the *Program.cs* file to the blob container. (This file is used here as an example; you could upload any text file and create a queue message with the file's name.)
 - In **Server Explorer**, double-click the node for the container you created.
 - In the **Container** window, select the **Upload** button.



- Find and select *Program.cs*, and then select **OK**.
- Create a queue message in the queue you created earlier, with *Program.cs* as the text of the message.



6. Run the project locally.

The queue message triggers the function, which then reads the blob and logs its length. The console output looks like this:

```
Found the following functions:  
ConsoleApp1.Functions.ProcessQueueMessage  
Job host started  
Executing 'Functions.ProcessQueueMessage' (Reason='New queue message detected on 'queue'.',  
Id=5a2ac479-de13-4f41-AAE9-1361f291ff88)  
Blob name:Program.cs  
Size: 532 bytes  
Executed 'Functions.ProcessQueueMessage' (Succeeded, Id=5a2ac479-de13-4f41-AAE9-1361f291ff88)
```

Add an output binding

Output bindings simplify code that writes data. This example modifies the previous one by writing a copy of the blob instead of logging its size. Blob storage bindings are included in the Azure Storage extension package that we installed previously.

1. Replace the `ProcessQueueMessage` method with the following code:

```
public static void ProcessQueueMessage(  
    [QueueTrigger("queue")] string message,  
    [Blob("container/{queueTrigger}", FileAccess.Read)] Stream myBlob,  
    [Blob("container/copy-{queueTrigger}", FileAccess.Write)] Stream outputBlob,  
    ILogger logger)  
{  
    logger.LogInformation($"Blob name:{message} \n Size: {myBlob.Length} bytes");  
    myBlob.CopyTo(outputBlob);  
}
```

2. Create another queue message with *Program.cs* as the text of the message.

3. Run the project locally.

The queue message triggers the function, which then reads the blob, logs its length, and creates a new blob. The console output is the same, but when you go to the blob container window and select **Refresh**, you see a new blob named *copy-Program.cs*.

Republish the updates to Azure

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog, make sure that the current profile is selected and then choose **Publish**. Results of the publish are detailed in the **Output** window.
3. Verify the function in Azure by again uploading a file to the blob container and adding a message to the queue that is the name of the uploaded file. You see the message get removed from the queue and a copy of the file created in the blob container.

Next steps

This article showed you how to create, run, and deploy a WebJobs SDK 3.x project.

[Learn more about the WebJobs SDK](#)

How to use the Azure WebJobs SDK for event-driven background processing

2/25/2020 • 25 minutes to read • [Edit Online](#)

This article provides guidance on how to work with the Azure WebJobs SDK. To get started with WebJobs right away, see [Get started with the Azure WebJobs SDK for event-driven background processing](#).

WebJobs SDK versions

These are the key differences between version 3.x and version 2.x of the WebJobs SDK:

- Version 3.x adds support for .NET Core.
- In version 3.x, you need to explicitly install the Storage binding extension required by the WebJobs SDK. In version 2.x, the Storage bindings were included in the SDK.
- Visual Studio tooling for .NET Core (3.x) projects differs from tooling for .NET Framework (2.x) projects. To learn more, see [Develop and deploy WebJobs using Visual Studio - Azure App Service](#).

When possible, examples are provided for both version 3.x and version 2.x.

NOTE

Azure Functions is built on the WebJobs SDK, and this article provides links to Azure Functions documentation for some topics. Note these differences between Functions and the WebJobs SDK:

- Azure Functions version 2.x corresponds to WebJobs SDK version 3.x, and Azure Functions 1.x corresponds to WebJobs SDK 2.x. Source code repositories use the WebJobs SDK numbering.
- Sample code for Azure Functions C# class libraries is like WebJobs SDK code, except you don't need a `FunctionName` attribute in a WebJobs SDK project.
- Some binding types are supported only in Functions, like HTTP (Webhooks) and Event Grid (which is based on HTTP).

For more information, see [Compare the WebJobs SDK and Azure Functions](#).

WebJobs host

The host is a runtime container for functions. It listens for triggers and calls functions. In version 3.x, the host is an implementation of `IHost`. In version 2.x, you use the `JobHost` object. You create a host instance in your code and write code to customize its behavior.

This is a key difference between using the WebJobs SDK directly and using it indirectly through Azure Functions. In Azure Functions, the service controls the host, and you can't customize the host by writing code. Azure Functions lets you customize host behavior through settings in the `host.json` file. Those settings are strings, not code, and this limits the kinds of customizations you can do.

Host connection strings

The WebJobs SDK looks for Azure Storage and Azure Service Bus connection strings in the `local.settings.json` file when you run locally, or in the environment of the WebJob when you run in Azure. By default, a storage connection string setting named `AzureWebJobsStorage` is required.

Version 2.x of the SDK lets you use your own names for these connection strings or store them elsewhere. You can set names in code using the `JobHostConfiguration`, as shown here:

```

static void Main(string[] args)
{
    var _storageConn = ConfigurationManager
        .ConnectionStrings["MyStorageConnection"].ConnectionString;

    //// Dashboard logging is deprecated; use Application Insights.
    //var _dashboardConn = ConfigurationManager
    //    .ConnectionStrings["MyDashboardConnection"].ConnectionString;

    JobHostConfiguration config = new JobHostConfiguration();
    config.StorageConnectionString = _storageConn;
    //config.DashboardConnectionString = _dashboardConn;
    JobHost host = new JobHost(config);
    host.RunAndBlock();
}

```

Because version 3.x uses the default .NET Core configuration APIs, there is no API to change connection string names.

Host development settings

You can run the host in development mode to make local development more efficient. Here are some of the settings that are changed when you run in development mode:

PROPERTY	DEVELOPMENT SETTING
Tracing.ConsoleLevel	TraceLevel.Verbose to maximize log output.
Queues.MaxPollingInterval	A low value to ensure queue methods are triggered immediately.
Singleton.ListenerLockPeriod	15 seconds to aid in rapid iterative development.

The process for enabling development mode depends on the SDK version.

Version 3.x

Version 3.x uses the standard ASP.NET Core APIs. Call the `UseEnvironment` method on the `HostBuilder` instance. Pass a string named `development`, as in this example:

```

static async Task Main()
{
    var builder = new HostBuilder();
    builder.UseEnvironment("development");
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}

```

Version 2.x

The `JobHostConfiguration` class has a `UseDevelopmentSettings` method that enables development mode. The following example shows how to use development settings. To make `config.IsDevelopment` return `true` when it runs locally, set a local environment variable named `AzureWebJobsEnv` with the value `Development`.

```

static void Main()
{
    config = new JobHostConfiguration();

    if (config.IsDevelopment)
    {
        config.UseDevelopmentSettings();
    }

    var host = new JobHost(config);
    host.RunAndBlock();
}

```

Managing concurrent connections (version 2.x)

In version 3.x, the connection limit defaults to infinite connections. If for some reason you need to change this limit, you can use the `MaxConnectionsPerServer` property of the `WinHttpHandler` class.

In version 2.x, you control the number of concurrent connections to a host by using the `ServicePointManager.DefaultConnectionLimit` API. In 2.x, you should increase this value from the default of 2 before starting your WebJobs host.

All outgoing HTTP requests that you make from a function by using `HttpClient` flow through `ServicePointManager`. After you reach the value set in `DefaultConnectionLimit`, `ServicePointManager` starts queueing requests before sending them. Suppose your `DefaultConnectionLimit` is set to 2 and your code makes 1,000 HTTP requests. Initially, only two requests are allowed through to the OS. The other 998 are queued until there's room for them. That means your `HttpClient` might time out because it appears to have made the request, but the request was never sent by the OS to the destination server. So you might see behavior that doesn't seem to make sense: your local `HttpClient` is taking 10 seconds to complete a request, but your service is returning every request in 200 ms.

The default value for ASP.NET applications is `Int32.MaxValue`, and that's likely to work well for WebJobs running in a Basic or higher App Service Plan. WebJobs typically need the Always On setting, and that's supported only by Basic and higher App Service Plans.

If your WebJob is running in a Free or Shared App Service Plan, your application is restricted by the App Service sandbox, which currently has a [connection limit of 300](#). With an unbound connection limit in `ServicePointManager`, it's more likely that the sandbox connection threshold will be reached and the site will shut down. In that case, setting `DefaultConnectionLimit` to something lower, like 50 or 100, can prevent this from happening and still allow for sufficient throughput.

The setting must be configured before any HTTP requests are made. For this reason, the WebJobs host shouldn't adjust the setting automatically. There could be HTTP requests that occur before the host starts, which could lead to unexpected behavior. The best approach is to set the value immediately in your `Main` method before initializing `JobHost`, as shown here:

```

static void Main(string[] args)
{
    // Set this immediately so that it's used by all requests.
    ServicePointManager.DefaultConnectionLimit = Int32.MaxValue;

    var host = new JobHost();
    host.RunAndBlock();
}

```

Triggers

Functions must be public methods and must have one trigger attribute or the `NoAutomaticTrigger` attribute.

Automatic triggers

Automatic triggers call a function in response to an event. Consider this example of a function that's triggered by a message added to Azure Queue storage. It responds by reading a blob from Azure Blob storage:

```
public static void Run(
    [QueueTrigger("myqueue-items")] string myQueueItem,
    [Blob("samples-workitems/{myQueueItem}", FileAccess.Read)] Stream myBlob,
    ILogger log)
{
    log.LogInformation($"BlobInput processed blob\n Name:{myQueueItem} \n Size: {myBlob.Length} bytes");
}
```

The `QueueTrigger` attribute tells the runtime to call the function whenever a queue message appears in the `myqueue-items` queue. The `Blob` attribute tells the runtime to use the queue message to read a blob in the `sample-workitems` container. The content of the queue message, passed in to the function in the `myQueueItem` parameter, is the name of the blob.

NOTE

A web app can time out after 20 minutes of inactivity. Only requests to the actual web app reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (https://<app_name>.scm.azurewebsites.net) don't reset the timer. If your app runs continuous or scheduled (Timer trigger) WebJobs, enable **Always On** to ensure that the WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium [pricing tiers](#).

Manual triggers

To trigger a function manually, use the `NoAutomaticTrigger` attribute, as shown here:

```
[NoAutomaticTrigger]
public static void CreateQueueMessage(
    ILogger logger,
    string value,
    [Queue("outputqueue")] out string message)
{
    message = value;
    logger.LogInformation("Creating queue message: ", message);
}
```

The process for manually triggering the function depends on the SDK version.

Version 3.x

```

static async Task Main(string[] args)
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage();
    });
    var host = builder.Build();
    using (host)
    {
        var jobHost = host.Services.GetService(typeof(IJobHost)) as JobHost;
        var inputs = new Dictionary<string, object>
        {
            { "value", "Hello world!" }
        };

        await host.StartAsync();
        await jobHost.CallAsync("CreateQueueMessage", inputs);
        await host.StopAsync();
    }
}

```

Version 2.x

```

static void Main(string[] args)
{
    JobHost host = new JobHost();
    host.Call(typeof(Program).GetMethod("CreateQueueMessage"), new { value = "Hello world!" });
}

```

Input and output bindings

Input bindings provide a declarative way to make data from Azure or third-party services available to your code. Output bindings provide a way to update data. The [Get started](#) article shows an example of each.

You can use a method return value for an output binding by applying the attribute to the method return value. See the example in [Using the Azure Function return value](#).

Binding types

The process for installing and managing binding types depends on whether you're using version 3x or version 2.x of the SDK. You can find the package to install for a particular binding type in the "Packages" section of that binding type's Azure Functions [reference article](#). An exception is the Files trigger and binding (for the local file system), which isn't supported by Azure Functions.

Version 3.x

In version 3.x, the storage bindings are included in the `Microsoft.Azure.WebJobs.Extensions.Storage` package. Call the `AddAzureStorage` extension method in the `ConfigureWebJobs` method, as shown here:

```

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}

```

To use other trigger and binding types, install the NuGet package that contains them and call the `Add<binding>` extension method implemented in the extension. For example, if you want to use an Azure Cosmos DB binding, install `Microsoft.Azure.WebJobs.Extensions.CosmosDB` and call `AddCosmosDB`, like this:

```

static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddCosmosDB();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}

```

To use the Timer trigger or the Files binding, which are part of core services, call the `AddTimers` or `AddFiles` extension methods, respectively.

Version 2.x

These trigger and binding types are included in version 2.x of the `Microsoft.Azure.WebJobs` package:

- Blob storage
- Queue storage
- Table storage

To use other trigger and binding types, install the NuGet package that contains them and call a `use<binding>` method on the `JobHostConfiguration` object. For example, if you want to use a Timer trigger, install `Microsoft.Azure.WebJobs.Extensions` and call `useTimers` in the `Main` method, as shown here:

```

static void Main()
{
    config = new JobHostConfiguration();
    config.UseTimers();
    var host = new JobHost(config);
    host.RunAndBlock();
}

```

To use the Files binding, install `Microsoft.Azure.WebJobs.Extensions` and call `UseFiles`.

ExecutionContext

WebJobs lets you bind to an `ExecutionContext`. With this binding, you can access the `ExecutionContext` as a parameter in your function signature. For example, the following code uses the context object to access the invocation ID, which you can use to correlate all logs produced by a given function invocation.

```
public class Functions
{
    public static void ProcessQueueMessage([QueueTrigger("queue")] string message,
        ExecutionContext executionContext,
        ILogger logger)
    {
        logger.LogInformation($"{message}\n{executionContext.InvocationId}");
    }
}
```

The process for binding to the `ExecutionContext` depends on your SDK version.

Version 3.x

Call the `AddExecutionContextBinding` extension method in the `ConfigureWebJobs` method, as shown here:

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddExecutionContextBinding();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

Version 2.x

The `Microsoft.Azure.WebJobs.Extensions` package mentioned earlier also provides a special binding type that you can register by calling the `UseCore` method. This binding lets you define an `ExecutionContext` parameter in your function signature, which is enabled like this:

```
class Program
{
    static void Main()
    {
        config = new JobHostConfiguration();
        config.UseCore();
        var host = new JobHost(config);
        host.RunAndBlock();
    }
}
```

Binding configuration

You can configure the behavior of some triggers and bindings. The process for configuring them depends on the SDK version.

- **Version 3.x:** Set configuration when the `Add<Binding>` method is called in `ConfigureWebJobs`.
- **Version 2.x:** Set configuration by setting properties in a configuration object that you pass in to `JobHost`.

These binding-specific settings are equivalent to settings in the [host.json project file](#) in Azure Functions.

You can configure the following bindings:

- [Azure CosmosDB trigger](#)
- [Event Hubs trigger](#)
- [Queue storage trigger](#)
- [SendGrid binding](#)
- [Service Bus trigger](#)

Azure CosmosDB trigger configuration (version 3.x)

This example shows how to configure the Azure Cosmos DB trigger:

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddCosmosDB(a =>
        {
            a.ConnectionMode = ConnectionMode.Gateway;
            a.Protocol = Protocol.Https;
            a.LeaseOptions.LeasePrefix = "prefix1";

        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more details, see the [Azure CosmosDB binding](#) article.

Event Hubs trigger configuration (version 3.x)

This example shows how to configure the Event Hubs trigger:

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddEventHubs(a =>
        {
            a.BatchCheckpointFrequency = 5;
            a.EventProcessorOptions.MaxBatchSize = 256;
            a.EventProcessorOptions.PrefetchCount = 512;
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more details, see the [Event Hubs binding](#) article.

Queue storage trigger configuration

These examples show how to configure the Queue storage trigger:

Version 3.x

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddAzureStorage(a => {
            a.BatchSize = 8;
            a.NewBatchThreshold = 4;
            a.MaxDequeueCount = 4;
            a.MaxPollingInterval = TimeSpan.FromSeconds(15);
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more details, see the [Queue storage binding](#) article.

Version 2.x

```
static void Main(string[] args)
{
    JobHostConfiguration config = new JobHostConfiguration();
    config.Queues.BatchSize = 8;
    config.Queues.NewBatchThreshold = 4;
    config.Queues.MaxDequeueCount = 4;
    config.Queues.MaxPollingInterval = TimeSpan.FromSeconds(15);
    JobHost host = new JobHost(config);
    host.RunAndBlock();
}
```

For more details, see the [host.json v1.x reference](#).

SendGrid binding configuration (version 3.x)

This example shows how to configure the SendGrid output binding:

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddSendGrid(a =>
        {
            a.FromAddress.Email = "samples@functions.com";
            a.FromAddress.Name = "Azure Functions";
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more details, see the [SendGrid binding](#) article.

Service Bus trigger configuration (version 3.x)

This example shows how to configure the Service Bus trigger:

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddServiceBus(sbOptions =>
        {
            sbOptions.MessageHandlerOptions.AutoComplete = true;
            sbOptions.MessageHandlerOptions.MaxConcurrentCalls = 16;
        });
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

For more details, see the [Service Bus binding](#) article.

Configuration for other bindings

Some trigger and binding types define their own custom configuration types. For example, the File trigger lets you specify the root path to monitor, as in these examples:

Version 3.x

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
        b.AddFiles(a => a.RootPath = @"c:\data\import");
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

Version 2.x

```
static void Main()
{
    config = new JobHostConfiguration();
    var filesConfig = new FilesConfiguration
    {
        RootPath = @"c:\data\import"
    };
    config.UseFiles(filesConfig);
    var host = new JobHost(config);
    host.RunAndBlock();
}
```

Binding expressions

In attribute constructor parameters, you can use expressions that resolve to values from various sources. For example, in the following code, the path for the `BlobTrigger` attribute creates an expression named `filename`. When used for the output binding, `filename` resolves to the name of the triggering blob.

```
public static void CreateThumbnail(
    [BlobTrigger("sample-images/{filename}")] Stream image,
    [Blob("sample-images-sm/{filename}", FileAccess.Write)] Stream imageSmall,
    string filename,
    ILogger logger)
{
    logger.Info($"Blob trigger processing: {filename}");
    // ...
}
```

For more information about binding expressions, see [Binding expressions and patterns](#) in the Azure Functions documentation.

Custom binding expressions

Sometimes you want to specify a queue name, a blob name or container, or a table name in code rather than hard-coding it. For example, you might want to specify the queue name for the `QueueTrigger` attribute in a configuration file or environment variable.

You can do that by passing a `NameResolver` object in to the `JobHostConfiguration` object. You include placeholders in trigger or binding attribute constructor parameters, and your `NameResolver` code provides the actual values to be used in place of those placeholders. You identify placeholders by surrounding them with percent (%) signs, as shown here:

```
public static void WriteLog([QueueTrigger("%logqueue%")] string logMessage)
{
    Console.WriteLine(logMessage);
}
```

This code lets you use a queue named `logqueue` in the test environment and one named `logqueueprod` in production. Instead of a hard-coded queue name, you specify the name of an entry in the `appSettings` collection.

There's a default `NameResolver` that takes effect if you don't provide a custom one. The default gets values from app settings or environment variables.

Your `NameResolver` class gets the queue name from `appSettings`, as shown here:

```
public class CustomNameResolver : INameResolver
{
    public string Resolve(string name)
    {
        return ConfigurationManager.AppSettings[name].ToString();
    }
}
```

Version 3.x

You configure the resolver by using dependency injection. These samples require the following `using` statement:

```
using Microsoft.Extensions.DependencyInjection;
```

You add the resolver by calling the `ConfigureServices` extension method on `HostBuilder`, as in this example:

```

static async Task Main(string[] args)
{
    var builder = new HostBuilder();
    var resolver = new CustomNameResolver();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureServices(s => s.AddSingleton<INameResolver>(resolver));
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}

```

Version 2.x

Pass your `NameResolver` class in to the `JobHost` object, as shown here:

```

static void Main(string[] args)
{
    JobHostConfiguration config = new JobHostConfiguration();
    config.NameResolver = new CustomNameResolver();
    JobHost host = new JobHost(config);
    host.RunAndBlock();
}

```

Azure Functions implements `INameResolver` to get values from app settings, as shown in the example. When you use the WebJobs SDK directly, you can write a custom implementation that gets placeholder replacement values from whatever source you prefer.

Binding at runtime

If you need to do some work in your function before you use a binding attribute like `Queue`, `Blob`, or `Table`, you can use the `IBinder` interface.

The following example takes an input queue message and creates a new message with the same content in an output queue. The output queue name is set by code in the body of the function.

```

public static void CreateQueueMessage(
    [QueueTrigger("inputqueue")] string queueMessage,
    IBinder binder)
{
    string outputQueueName = "outputqueue" + DateTime.Now.Month.ToString();
    QueueAttribute queueAttribute = new QueueAttribute(outputQueueName);
    CloudQueue outputQueue = binder.Bind<CloudQueue>(queueAttribute);
    outputQueue.AddMessageAsync(new CloudQueueMessage(queueMessage));
}

```

For more information, see [Binding at runtime](#) in the Azure Functions documentation.

Binding reference information

The Azure Functions documentation provides reference information about each binding type. You'll find the following information in each binding reference article. (This example is based on Storage queue.)

- [Packages](#). The package you need to install to include support for the binding in a WebJobs SDK project.
- [Examples](#). Code samples. The C# class library example applies to the WebJobs SDK. Just omit the

`FunctionName` attribute.

- [Attributes](#). The attributes to use for the binding type.
- [Configuration](#). Explanations of the attribute properties and constructor parameters.
- [Usage](#). The types you can bind to and information about how the binding works. For example: polling algorithm, poison queue processing.

For a list of binding reference articles, see "Supported bindings" in the [Triggers and bindings](#) article for Azure Functions. In that list, the HTTP, Webhooks, and Event Grid bindings are supported only by Azure Functions, not by the WebJobs SDK.

Disable attribute

The `Disable` attribute lets you control whether a function can be triggered.

In the following example, if the app setting `Disable_TestJob` has a value of `1` or `True` (case insensitive), the function won't run. In that case, the runtime creates a log message *Function 'Functions.TestJob' is disabled*.

```
[Disable("Disable_TestJob")]
public static void TestJob([QueueTrigger("testqueue2")] string message)
{
    Console.WriteLine("Function with Disable attribute executed!");
}
```

When you change app setting values in the Azure portal, the WebJob restarts to pick up the new setting.

The attribute can be declared at the parameter, method, or class level. The setting name can also contain binding expressions.

Timeout attribute

The `Timeout` attribute causes a function to be canceled if it doesn't finish within a specified amount of time. In the following example, the function would run for one day without the Timeout attribute. Timeout causes the function to be canceled after 15 seconds.

```
[Timeout("00:00:15")]
public static async Task TimeoutJob(
    [QueueTrigger("testqueue2")] string message,
    CancellationToken token,
    TextWriter log)
{
    await log.WriteLineAsync("Job starting");
    await Task.Delay(TimeSpan.FromDays(1), token);
    await log.WriteLineAsync("Job completed");
}
```

You can apply the Timeout attribute at the class or method level, and you can specify a global timeout by using `JobHostConfiguration.FunctionTimeout`. Class-level or method-level timeouts override global timeouts.

Singleton attribute

The `Singleton` attribute ensures that only one instance of a function runs, even when there are multiple instances of the host web app. It does this by using [distributed locking](#).

In this example, only a single instance of the `ProcessImage` function runs at any given time:

```
[Singleton]
public static async Task ProcessImage([BlobTrigger("images")] Stream image)
{
    // Process the image.
}
```

SingletonMode.Listener

Some triggers have built-in support for concurrency management:

- **QueueTrigger**. Set `JobHostConfiguration.Queues.BatchSize` to `1`.
- **ServiceBusTrigger**. Set `ServiceBusConfiguration.MessageOptions.MaxConcurrentCalls` to `1`.
- **FileTrigger**. Set `FileProcessor.MaxDegreeOfParallelism` to `1`.

You can use these settings to ensure that your function runs as a singleton on a single instance. To ensure that only a single instance of the function is running when the web app scales out to multiple instances, apply a listener-level singleton lock on the function (`[singleton(mode = SingletonMode.Listener)]`). Listener locks are acquired when the JobHost starts. If three scaled-out instances all start at the same time, only one of the instances acquires the lock and only one listener starts.

Scope values

You can specify a *scope expression/value* on a singleton. The expression/value ensures that all executions of the function at a specific scope will be serialized. Implementing more granular locking in this way can allow for some level of parallelism for your function while serializing other invocations as dictated by your requirements. For example, in the following code, the scope expression binds to the `Region` value of the incoming message. When the queue contains three messages in regions East, East, and West respectively, the messages that have region East are run serially while the message with region West is run in parallel with those in East.

```
[Singleton("{Region}")]
public static async Task ProcessWorkItem([QueueTrigger("workitems")] WorkItem workItem)
{
    // Process the work item.
}

public class WorkItem
{
    public int ID { get; set; }
    public string Region { get; set; }
    public int Category { get; set; }
    public string Description { get; set; }
}
```

SingletonScope.Host

The default scope for a lock is `SingletonScope.Function`, meaning the lock scope (the blob lease path) is tied to the fully qualified function name. To lock across functions, specify `SingletonScope.Host` and use a scope ID name that's the same across all functions that you don't want to run simultaneously. In the following example, only one instance of `AddItem` or `RemoveItem` runs at a time:

```

[Singleton("ItemsLock", SingletonScope.Host)]
public static void AddItem([QueueTrigger("add-item")] string message)
{
    // Perform the add operation.
}

[Singleton("ItemsLock", SingletonScope.Host)]
public static void RemoveItem([QueueTrigger("remove-item")] string message)
{
    // Perform the remove operation.
}

```

Viewing lease blobs

The WebJobs SDK uses [Azure blob leases](#) under the covers to implement distributed locking. The lease blobs used by Singleton can be found in the `azure-webjobs-host` container in the `AzureWebJobsStorage` storage account under the path "locks". For example, the lease blob path for the first `ProcessImage` example shown earlier might be `locks/061851c758f04938a4426aa9ab3869c0/WebJobs.Functions.ProcessImage`. All paths include the JobHost ID, in this case 061851c758f04938a4426aa9ab3869c0.

Async functions

For information about how to code async functions, see the [Azure Functions documentation](#).

Cancellation tokens

For information about how to handle cancellation tokens, see the Azure Functions documentation on [cancellation tokens and graceful shutdown](#).

Multiple instances

If your web app runs on multiple instances, a continuous WebJob runs on each instance, listening for triggers and calling functions. The various trigger bindings are designed to efficiently share work collaboratively across instances, so that scaling out to more instances allows you to handle more load.

While some triggers may result in double-processing, queue and blob storage triggers automatically prevent a function from processing a queue message or blob more than once. For more information, see [Designing for identical input](#) in the Azure Functions documentation.

The timer trigger automatically ensures that only one instance of the timer runs, so you don't get more than one function instance running at a given scheduled time.

If you want to ensure that only one instance of a function runs even when there are multiple instances of the host web app, you can use the `Singleton` attribute.

Filters

Function Filters (preview) provide a way to customize the WebJobs execution pipeline with your own logic. Filters are similar to [ASP.NET Core filters](#). You can implement them as declarative attributes that are applied to your functions or classes. For more information, see [Function Filters](#).

Logging and monitoring

We recommend the logging framework that was developed for ASP.NET. The [Get started](#) article shows how to use it.

Log filtering

Every log created by an `ILogger` instance has an associated `Category` and `Level`. `LogLevel` is an enumeration, and the integer code indicates relative importance:

LOGLEVEL	CODE
Trace	0
Debug	1
Information	2
Warning	3
Error	4
Critical	5
None	6

You can independently filter each category to a particular `LogLevel`. For example, you might want to see all logs for blob trigger processing but only `Error` and higher for everything else.

Version 3.x

Version 3.x of the SDK relies on the filtering built into .NET Core. The `LogCategories` class lets you define categories for specific functions, triggers, or users. It also defines filters for specific host states, like `Startup` and `Results`. This enables you to fine-tune the logging output. If no match is found within the defined categories, the filter falls back to the `Default` value when deciding whether to filter the message.

`LogCategories` requires the following using statement:

```
using Microsoft.Azure.WebJobs.Logging;
```

The following example constructs a filter that, by default, filters all logs at the `Warning` level. The `Function` and `results` categories (equivalent to `Host.Results` in version 2.x) are filtered at the `Error` level. The filter compares the current category to all registered levels in the `LogCategories` instance and chooses the longest match. This means that the `Debug` level registered for `Host.Triggers` matches `Host.Triggers.Queue` or `Host.Triggers.Blob`. This allows you to control broader categories without needing to add each one.

```

static async Task Main(string[] args)
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureLogging(logging =>
    {
        logging.SetMinimumLevel(LogLevel.Warning);
        logging.AddFilter("Function", LogLevel.Error);
        logging.AddFilter(LogCategories.CreateFunctionCategory("MySpecificFunctionName"),
            LogLevel.Debug);
        logging.AddFilter(LogCategories.Results, LogLevel.Error);
        logging.AddFilter("Host.Triggers", LogLevel.Debug);
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}

```

Version 2.x

In version 2.x of the SDK, you use the `LogCategoryFilter` class to control filtering. The `LogCategoryFilter` has a `Default` property with an initial value of `Information`, meaning that any messages at the `Information`, `Warning`, `Error`, or `Critical` levels are logged, but any messages at the `Debug` or `Trace` levels are filtered away.

As with `LogCategories` in version 3.x, the `CategoryLevels` property allows you to specify log levels for specific categories so you can fine-tune the logging output. If no match is found within the `CategoryLevels` dictionary, the filter falls back to the `Default` value when deciding whether to filter the message.

The following example constructs a filter that by default filters all logs at the `Warning` level. The `Function` and `Host.Results` categories are filtered at the `Error` level. The `LogCategoryFilter` compares the current category to all registered `CategoryLevels` and chooses the longest match. So the `Debug` level registered for `Host.Triggers` will match `Host.Triggers.Queue` or `Host.Triggers.Blob`. This allows you to control broader categories without needing to add each one.

```

var filter = new LogCategoryFilter();
filter.DefaultLevel = LogLevel.Warning;
filter.CategoryLevels[LogCategories.Function] = LogLevel.Error;
filter.CategoryLevels[LogCategories.Results] = LogLevel.Error;
filter.CategoryLevels["Host.Triggers"] = LogLevel.Debug;

config.LoggerFactory = new LoggerFactory()
    .AddApplicationInsights(instrumentationKey, filter.Filter)
    .AddConsole(filter.Filter);

```

Custom telemetry for Application Insights

The process for implementing custom telemetry for [Application Insights](#) depends on the SDK version. To learn how to configure Application Insights, see [Add Application Insights logging](#).

Version 3.x

Because version 3.x of the WebJobs SDK relies on the .NET Core generic host, a custom telemetry factory is no longer provided. But you can add custom telemetry to the pipeline by using dependency injection. The examples in this section require the following `using` statements:

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.ApplicationInsights.Channel;
```

The following custom implementation of `ITelemetryInitializer` lets you add your own `ITelemetry` to the default `TelemetryConfiguration`.

```
internal class CustomTelemetryInitializer : ITelemetryInitializer
{
    public void Initialize(ITelemetry telemetry)
    {
        // Do something with telemetry.
    }
}
```

Call `ConfigureServices` in the builder to add your custom `ITelemetryInitializer` to the pipeline.

```
static async Task Main()
{
    var builder = new HostBuilder();
    builder.ConfigureWebJobs(b =>
    {
        b.AddAzureStorageCoreServices();
    });
    builder.ConfigureLogging((context, b) =>
    {
        // Add logging providers.
        b.AddConsole();

        // If this key exists in any config, use it to enable Application Insights.
        string appInsightsKey = context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
        if (!string.IsNullOrEmpty(appInsightsKey))
        {
            // This uses the options callback to explicitly set the instrumentation key.
            b.AddApplicationInsights(o => o.InstrumentationKey = appInsightsKey);
        }
    });
    builder.ConfigureServices(services =>
    {
        services.AddSingleton<ITelemetryInitializer, CustomTelemetryInitializer>();
    });
    var host = builder.Build();
    using (host)
    {
        await host.RunAsync();
    }
}
```

When the `TelemetryConfiguration` is constructed, all registered types of `ITelemetryInitializer` are included. To learn more, see [Application Insights API for custom events and metrics](#).

In version 3.x, you no longer have to flush the `TelemetryClient` when the host stops. The .NET Core dependency injection system automatically disposes of the registered `ApplicationInsightsLoggerProvider`, which flushes the `TelemetryClient`.

Version 2.x

In version 2.x, the `TelemetryClient` created internally by the Application Insights provider for the WebJobs SDK uses `ServerTelemetryChannel`. When the Application Insights endpoint is unavailable or throttling incoming requests, this channel [saves requests in the web app's file system and resubmits them later](#).

The `TelemetryClient` is created by a class that implements `ITelemetryClientFactory`. By default, this is the

```
DefaultTelemetryClientFactory .
```

If you want to modify any part of the Application Insights pipeline, you can supply your own `ITelemetryClientFactory`, and the host will use your class to construct a `TelemetryClient`. For example, this code overrides `DefaultTelemetryClientFactory` to modify a property of `ServerTelemetryChannel`:

```
private class CustomTelemetryClientFactory : DefaultTelemetryClientFactory
{
    public CustomTelemetryClientFactory(string instrumentationKey, Func<string, LogLevel, bool> filter)
        : base(instrumentationKey, new SamplingPercentageEstimatorSettings(), filter)
    {
    }

    protected override ITelemetryChannel CreateTelemetryChannel()
    {
        ServerTelemetryChannel channel = new ServerTelemetryChannel();

        // Change the default from 30 seconds to 15 seconds.
        channel.MaxTelemetryBufferDelay = TimeSpan.FromSeconds(15);

        return channel;
    }
}
```

The `SamplingPercentageEstimatorSettings` object configures [adaptive sampling](#). This means that in certain high-volume scenarios, Applications Insights sends a selected subset of telemetry data to the server.

After you create the telemetry factory, you pass it in to the Application Insights logging provider:

```
var clientFactory = new CustomTelemetryClientFactory(instrumentationKey, filter.Filter);

config.LoggerFactory = new LoggerFactory()
    .AddApplicationInsights(clientFactory);
```

Next steps

This article has provided code snippets that show how to handle common scenarios for working with the WebJobs SDK. For complete samples, see [azure-webjobs-sdk-samples](#).

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 ¹	Unlimited.	Unlimited.
Unique tag calculations per subscription ¹	10,000	10,000
Subscription-level deployments per location	800 ²	800

¹You 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.

²If 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 resource group	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 Resources not limited to 800 instances per resource group .
Deployments per resource group in the deployment history	800 ¹	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

¹If 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"> 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. 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 create and assign a custom role with permission to create a limitless number of app registrations.

CATEGORY	LIMITS
Schema extensions	<ul style="list-style-type: none"> String-type extensions can have a maximum of 256 characters. Binary-type extensions are limited to 256 bytes. Only 100 extension values, across <i>all</i> types and <i>all</i> applications, can be written to any single Azure AD resource. Only User, Group, TenantDetail, Device, Application, and ServicePrincipal entities can be extended with string-type or binary-type single-valued attributes. Schema extensions are available only in the Graph API version 1.21 preview. The application must be granted write access to register an extension.
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"> A user can create a maximum of 250 groups in an Azure AD organization. An Azure AD organization can have a maximum of 5000 dynamic groups. A maximum of 100 users can be owners of a single group. Any number of Azure AD resources can be members of a single group. A user can be a member of any number of groups. 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. Nested Groups in Azure AD are not supported within all scenarios <p>At this time the following are the supported scenarios with nested groups.</p> <ul style="list-style-type: none"> One group can be added as a member of another group and you can achieve group nesting. 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) Conditional access (when scoping a conditional access policy to a group) Restricting access to self-serve password reset Restricting which users can do Azure AD Join and device registration <p>The following scenarios DO NOT support nested groups:</p> <ul style="list-style-type: none"> 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 Group-based licensing (assigning a license automatically to all members of a group) Office 365 Groups.
Application Proxy	<ul style="list-style-type: none"> A maximum of 500 transactions per second per App Proxy application A maximum of 750 transactions per second for the Azure AD organization <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"> 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. 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.
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"> A group cannot be added as an owner. A group cannot be assigned to a role. 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 here. It may take up to 15 minutes or signing out/signing in before admin role membership additions and revocations take effect.

API Management limits

RESOURCE	LIMIT
Maximum number of scale units	10 per region ¹
Cache size	5 GiB per unit ²
Concurrent back-end connections ³ per HTTP authority	2,048 per unit ⁴
Maximum cached response size	2 MiB
Maximum policy document size	256 KiB ⁵
Maximum custom gateway domains per service instance ⁶	20
Maximum number of CA certificates per service instance	10
Maximum number of service instances per subscription ⁷	20
Maximum number of subscriptions per service instance ⁷	500
Maximum number of client certificates per service instance ⁷	50

RESOURCE	LIMIT
Maximum number of APIs per service instance ⁷	50
Maximum number of API operations per service instance ⁷	1,000
Maximum total request duration ⁷	30 seconds
Maximum buffered payload size ⁷	2 MiB
Maximum request URL size ⁸	4096 bytes

¹Scaling limits depend on the pricing tier. To see the pricing tiers and their scaling limits, see [API Management pricing](#).

²Per unit cache size depends on the pricing tier. To see the pricing tiers and their scaling limits, see [API Management pricing](#).

³Connections are pooled and reused unless explicitly closed by the back end.

⁴This 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.

⁵This limit applies to the Basic, Standard, and Premium tiers. In the Consumption tier, policy document size is limited to 4 KiB.

⁶This resource is available in the Premium tier only.

⁷This resource applies to the Consumption tier only.

⁸Applies 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 ¹	10	100	Unlimited ²	Unlimited ²	Unlimited ²	Unlimited ²
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 ³	Dedicated ³	Dedicated ³	Dedicated ³
Scale out (maximum instances)	1 shared	1 shared	3 dedicated ³	10 dedicated ³	30 dedicated ³	100 dedicated ⁴
Storage ⁵	1 GB ⁵	1 GB ⁵	10 GB ⁵	50 GB ⁵	250 GB ⁵	1 TB ⁵
CPU time (5 minutes) ⁶	3 minutes	3 minutes	Unlimited, pay at standard rates			

Resource	Free	Shared	Basic	Standard	Premium (V2)	Isolated
CPU time (day) ⁶	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 ⁷	5	35	350	Unlimited	Unlimited	Unlimited
IP connections	600	600	Depends on instance size ⁸	Depends on instance size ⁸	Depends on instance size ⁸	16,000
Concurrent debugger connections per application	1	1	1	5	5	5
App Service Certificates per subscription ⁹	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 ¹⁰
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 ¹¹	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%

¹Apps and storage quotas are per App Service plan unless noted otherwise.

²The 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.

³Dedicated instances can be of different sizes. For more information, see [App Service pricing](#).

⁴More are allowed upon request.

⁵The 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.

⁶These resources are constrained by physical resources on the dedicated instances (the instance size and the number of instances).

⁷If 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).

⁸The 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.

⁹The App Service Certificate quota limit per subscription can be increased via a support request to a maximum limit of 200.

¹⁰App 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.

¹¹Run 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 ¹	1 GB	Applies to Azure sandboxes only.
Maximum amount of memory given to a sandbox ¹	400 MB	Applies to Azure sandboxes only.
Maximum number of network sockets allowed per sandbox ¹	1,000	Applies to Azure sandboxes only.
Maximum runtime allowed per runbook ¹	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.

¹A 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 ¹	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

¹Each 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 ¹	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) ²	N/A	3 SU	36 SU	36 SU	36 SU	36 SU	36 SU	36 SU

¹ Free is based on shared, not dedicated, resources. Scale-up is not supported on shared resources.

² 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 ¹	S1	S2	S3	S3 HD ²	L1	L2
Service level agreement (SLA) ³	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

¹ Basic has one fixed partition. At this tier, additional search units are used for allocating more replicas for increased query workloads.

² 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.

³ 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 ¹
Scale out	Event driven	Event driven	Manual/autoscale

RESOURCE	CONSUMPTION PLAN	PREMIUM PLAN	APP SERVICE PLAN
Max instances	200	100	10-20
Default timeout duration (min)	5	30	30 ²
Max timeout duration (min)	10	unbounded ⁸	unbounded ³
Max outbound connections (per instance)	600 active (1200 total)	unbounded	unbounded
Max request size (MB) ⁴	100	100	100
Max query string length ⁴	4096	4096	4096
Max request URL length ⁴	8192	8192	8192
ACU 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 ⁵
App Service plans	100 per region	100 per resource group	100 per resource group
Storage ⁶	1 GB	250 GB	50-1000 GB
Custom domains per app	500 ⁷	500	500
Custom domain SSL support	unbounded SNI SSL connection included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included

¹ For specific limits for the various App Service plan options, see the [App Service plan limits](#).

² By default, the timeout for the Functions 1.x runtime in an App Service plan is unbounded.

³ Requires the App Service plan be set to [Always On](#). Pay at standard [rates](#).

⁴ These limits are [set in the host](#).

⁵ 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.

⁶ 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.

⁷ 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.

⁸ 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 Standard Load Balancer SKU	1000 (100 nodes per node pool)
Maximum pods per node: Basic networking with Kubenet	110
Maximum pods per node: Advanced networking with Azure Container Networking Interface	Azure CLI deployment: 30 ¹ Azure Resource Manager template: 30 ¹ Portal deployment: 30

¹When 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 rate limiting information .	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 rate limiting information .	Call support.
Voice	10 voice actions in an action group. No more than 1 voice call every 5 minutes. Also see the rate limiting information .	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 action-specific information .	Call support.

Log queries and language

LIMIT	DESCRIPTION
Query language	Azure Monitor uses the same Kusto query language as Azure Data Explorer. See Azure Monitor log query language differences 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 Query limits 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 Cross-resource query limits 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 Timeouts for details.
Maximum request rate	200 requests per 30 seconds per AAD user or client IP address	See Rate limits 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 AIDataCap@microsoft.com .
Throttling	32,000 events/second	The limit is measured over a minute.
Data retention	90 days	This resource is for Search , Analytics , and Metrics Explorer .
Availability multi-step test 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 type schemas .
Property value string length	8,192	See type schemas .
Trace and exception message length	32,768	See type schemas .
Availability tests 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
Active jobs and job schedules per Batch account (completed jobs have no limit)	100-300	1,000 ¹
Pools per Batch account	20-100	500 ¹

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](#).

¹To 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 subscription ¹	20	10,000
Coadministrators per subscription	200	200
Storage accounts per subscription ²	100	100
Cloud services per subscription	20	200
Local networks 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

¹Extra small instances count as one vCPU toward the vCPU limit despite using a partial CPU core.

²The storage account limit includes both Standard and Premium storage accounts.

Container Instances limits

RESOURCE	DEFAULT LIMIT
Standard sku container groups per region per subscription	100 ¹
Dedicated sku container groups per region per subscription	0 ¹
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 ¹
Container creates per 5 minutes	100 ¹
Container deletes per hour	300 ¹
Container deletes per 5 minutes	100 ¹

¹To 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 ¹	10 GiB	100 GiB	500 GiB
Maximum image layer size	200 GiB	200 GiB	200 GiB
ReadOps per minute ^{2, 3}	1,000	3,000	10,000
WriteOps per minute ^{2, 4}	100	500	2,000

RESOURCE	BASIC	STANDARD	PREMIUM
Download bandwidth MBps ²	30	60	100
Upload bandwidth MBps ²	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

¹The 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](#).

²*ReadOps*, *WriteOps*, and *Bandwidth* are minimum estimates. Azure Container Registry strives to improve performance as usage requires.

³A [docker pull](#) translates to multiple read operations based on the number of layers in the image, plus the manifest retrieval.

⁴A [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 Azure Integration Runtime region 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 Azure Integration Runtime region Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete.	1000	Contact support.
Concurrent authoring operations per subscription per Azure Integration Runtime region Including test connection, browse folder list and table list, preview data.	200	Contact support.
Concurrent Data Integration Units ¹ consumption per subscription per Azure Integration Runtime region	Region group 1 ² : 6000 Region group 2 ² : 3000 Region group 3 ² : 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 ³	200 KB	200 KB
Bytes per object for dataset and linked service objects ³	100 KB	2,000 KB
Data Integration Units ¹ per copy activity run	256	Contact support.
Write API calls	1,200/h This limit is imposed by Azure Resource Manager, not Azure Data Factory.	Contact support.
Read API calls	12,500/h This limit is imposed by Azure Resource Manager, not Azure Data Factory.	Contact support.
Monitoring queries per minute	1,000	Contact support.
Entity CRUD operations per minute	50	Contact support.
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per factory	50	Contact support.
Concurrent number of data flow debug sessions per user per factory	3	3
Data Flow Azure IR TTL limit	4 hrs	Contact support.

¹ 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](#).

² [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

³ 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	Contact support .
Data sets within a data factory	5,000	Contact support .
Concurrent slices per data set	10	10
Bytes per object for pipeline objects ¹	200 KB	200 KB
Bytes per object for data set and linked service objects ¹	100 KB	2,000 KB
Azure HDInsight on-demand cluster cores within a subscription ²	60	Contact support .
Cloud data movement units per copy activity run ³	32	Contact support .
Retry count for pipeline activity runs	1,000	MaxInt (32 bit)

¹ 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.

² 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.

³ 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 server busy exception . To request a larger number of throughput units for a Standard tier, file a support request . 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">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.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.

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
Maximum size of device twin	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
Maximum depth of object in device twin	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 region¹):

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 ¹
All transactions	2,000

For information on how to handle throttling when these limits are exceeded, see [Azure Key Vault throttling guidance](#).

¹ 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) ¹
Jobs per Media Services account	50,000 ²
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 ⁴
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 ⁵ (fixed)
Policies	1,000,000 ⁶
File size	In some scenarios, there's a limit on the maximum file size supported for processing in Media Services. ⁷

¹If you change the type, for example, from S2 to S1, the maximum reserved unit limits are reset.

²This 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](#).

³When 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](#).

⁴Locators 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](#).

⁵The storage accounts must be from the same Azure subscription.

⁶There'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](#).

⁷The 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 X attempts during MFA call	1	99
Two-way text message timeout seconds	60	600
Default one-time bypass seconds	300	1,800
Lock user account after X consecutive MFA denials	Not set	99
Reset account lockout counter after X minutes	Not set	9,999
Unlock account after X minutes	Not set	9,999

Networking limits

Networking limits - Azure Resource Manager The following limits apply only for networking resources managed through **Azure Resource Manager** per region per subscription. Learn how to [view your current resource usage against your subscription limits](#).

NOTE

We recently increased all default limits to their maximum limits. If there's no maximum limit column, the resource doesn't have adjustable limits. If you had these limits increased by support in the past and don't see updated limits in the following tables, [open an online customer support request at no charge](#)

RESOURCE	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 ¹	10 for Basic.	Contact support.
Static Public IP addresses ¹	10 for Basic.	Contact support.
Standard Public IP addresses ¹	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.

¹Default 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 Number of virtual networks per ExpressRoute circuit table .

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 ¹	
Back-end address pools	100 ¹	
Back-end servers per pool	1,200	
HTTP listeners	100 ¹	
HTTP load-balancing rules	100 ¹	
Back-end HTTP settings	100 ¹	
Instances per gateway	V1 SKU - 32 V2 SKU - 125	
SSL certificates	100 ¹	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 ¹	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 ¹	
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	

¹ 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 ¹
Record sets per public DNS zone	10,000 ¹
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 ²
Maximum number of DNS queries queued (pending response) per virtual machine	200 ²

¹If you need to increase these limits, contact Azure Support.

²These 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 ¹
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 NextHopType value set to Internet.</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 NextHopType value set as Internet 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>

¹If 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
Download	There's no limit on the download size.	There's no limit on the download size.

	WITH CHUNKED TRANSFER ENCODING (CTE)	WITHOUT HTTP CHUNKING
Upload	There's no limit as long as each CTE upload is less than 2 GB.	The size can't be larger than 2 GB.

Other limits

- Maximum URL size - 8,192 bytes - Specifies maximum length of the raw URL (scheme + hostname + port + path + query string of the URL)
- Maximum Query String size - 4,096 bytes - Specifies the maximum length of the query string, in bytes.

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 partitioning 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 Azure portal , 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 partitioned topics or queues 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 Azure portal , an error message is generated. If called from the management API, the exception QuotaExceededException 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 Standard tier, 1 MB for Premium tier.</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: byte/int.MaxValue.</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 SerializationException 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 BrokeredMessage , which has both user properties and system properties, such as SequenceNumber , Label , and MessageId .
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 SharedAccessAuthorizationRule 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 Send() and SendAsync() 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) ¹
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account	No limit
Maximum request rate ¹ per storage account	20,000 requests per second
Maximum ingress ¹ per storage account (US, Europe regions)	25 Gbps
Maximum ingress ¹ per storage account (regions other than US and Europe)	5 Gbps if RA-GRS/GRS is enabled, 10 Gbps for LRS/ZRS ²
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 ²
Maximum egress for general-purpose v1 storage accounts (non-US regions)	10 Gbps if RA-GRS/GRS is enabled, 15 Gbps for LRS/ZRS ²
Maximum number of virtual network rules per storage account	200
Maximum number of IP address rules per storage account	200

¹Azure 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](#).

²If 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 ¹

¹ 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 Property Types in Understanding the Table Service Data Model .
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) ¹	<=50 Gbps

¹Ingress 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"> If there are more than 16 volumes, they're processed sequentially as processing slots become available. 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.
Restore and clone recover time for tiered volumes	<2 minutes	<ul style="list-style-type: none"> The volume is made available within 2 minutes of a restore or clone operation, regardless of the volume size. 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. 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. The restore or clone operation is complete when all the metadata is on the device. Backup operations can't be performed until the restore or clone operation is fully complete.

LIMIT IDENTIFIER	LIMIT	COMMENTS
Restore recover time for locally pinned volumes	<2 minutes	<ul style="list-style-type: none"> The volume is made available within 2 minutes of the restore operation, regardless of the volume size. 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. 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. 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. 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.
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 Microsoft Support .
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
Virtual machines per cloud service ¹	50	50
Input endpoints per cloud service ²	150	150

¹Virtual 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.

²Input 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 subscription	25,000 ¹ per region.
VM total cores per subscription	20 ¹ per region. Contact support to increase limit.
Azure Spot VM total cores per subscription	20 ¹ per region. Contact support to increase limit.
VM per series, such as Dv2 and F, cores per subscription	20 ¹ 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 ²

¹Default 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.

²With 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](#)  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 connections 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

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**.

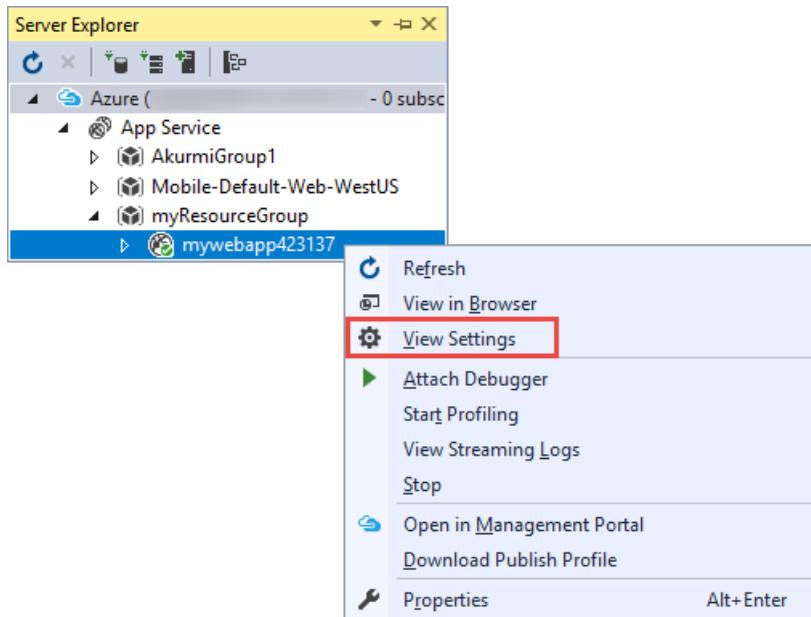
An alternative is to install a management certificate that enables access to your account. If you choose to install a certificate, right-click the **Azure** node in **Server Explorer**, and then select **Manage and Filter Subscriptions** in the context menu. In the **Manage Microsoft Azure Subscriptions** 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:

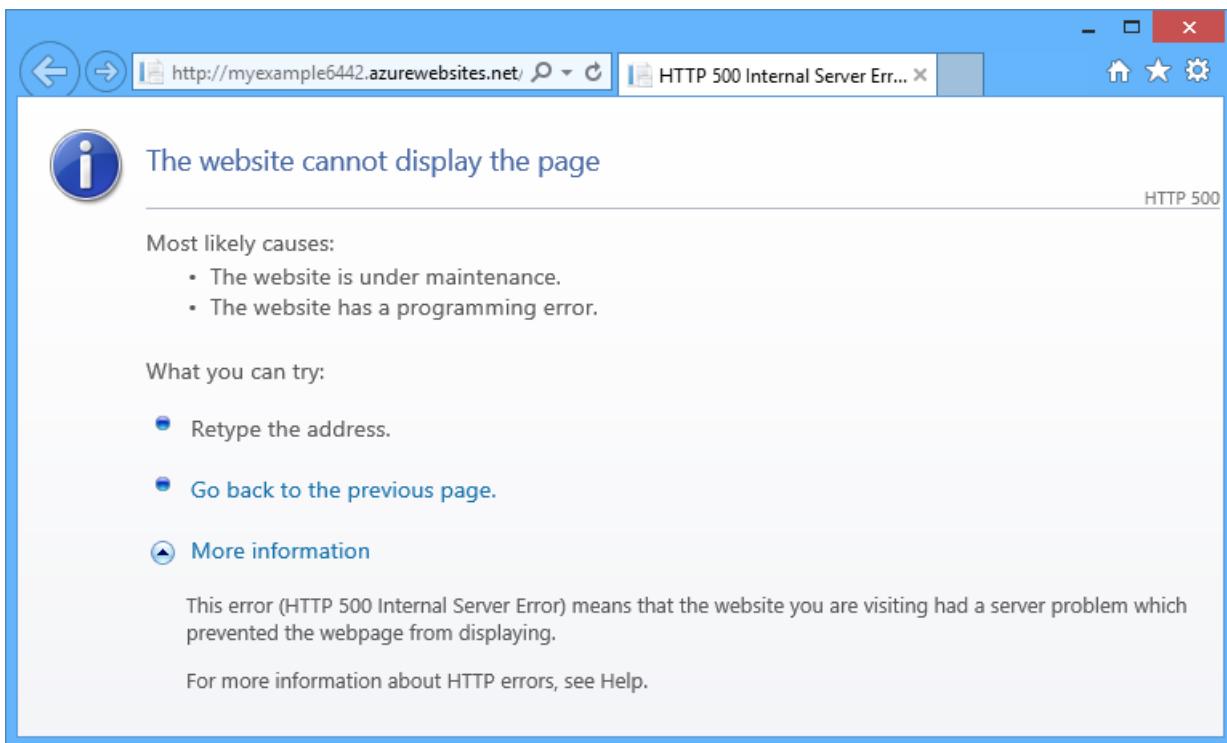
Application name

Error.

An error occurred while processing your request.

© 2014 - My ASP.NET Application

The website cannot display the page

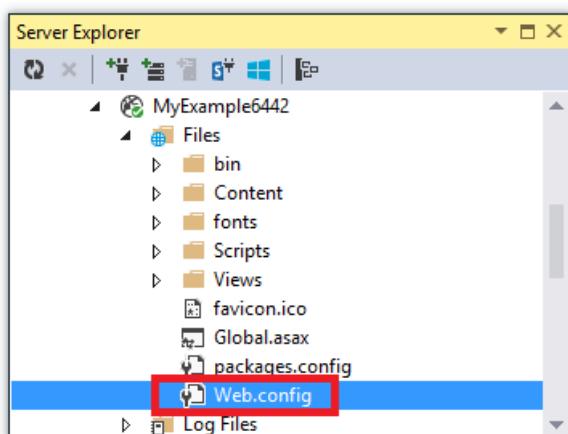


Frequently the easiest way to find the cause of the error is to enable detailed error messages, which the first of the preceding screenshots explains how to do. That requires a change in the deployed Web.config file. You could edit the `Web.config` file in the project and redeploy the project, or create 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.

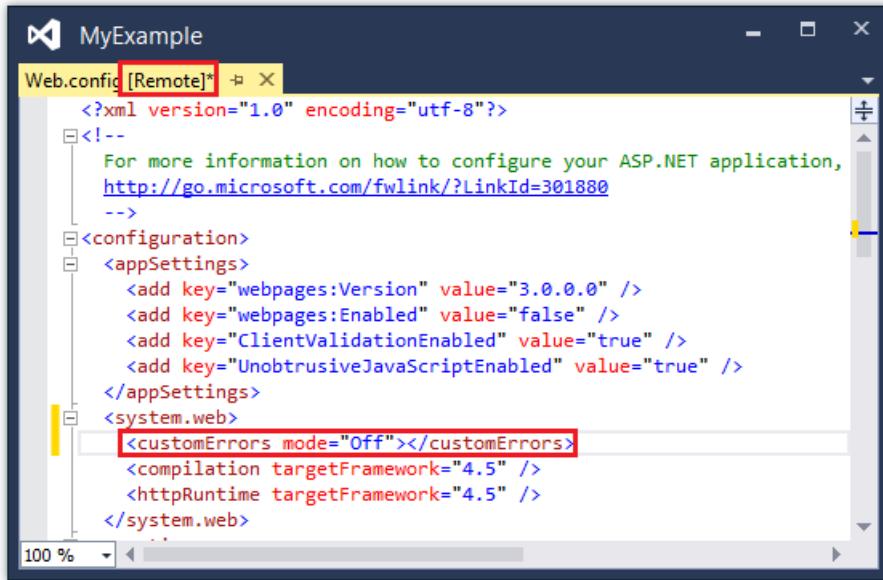
2. Expand the **Files** node, and double-click the `Web.config` file.



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.

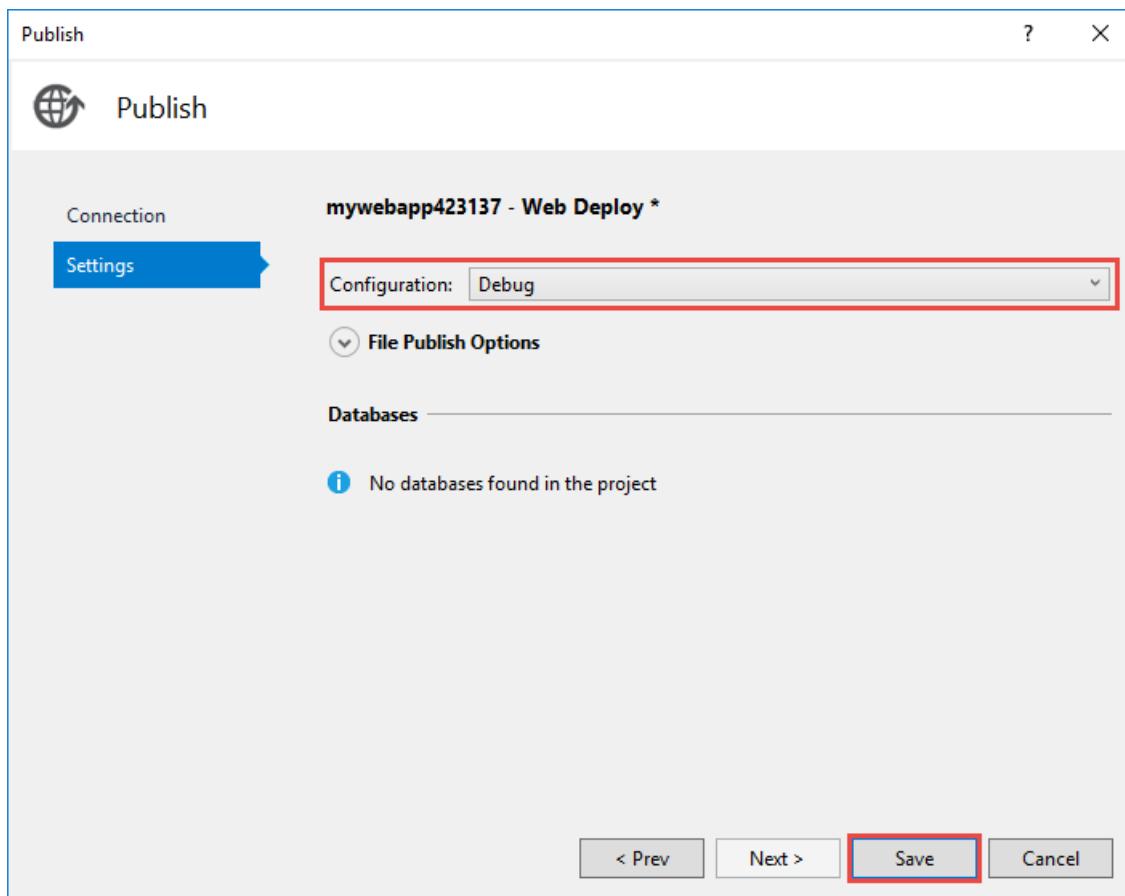
Remote debugging does not work in Express editions of Visual Studio.

This section shows how to debug remotely using the project you create in [Create an ASP.NET app in Azure App Service](#).

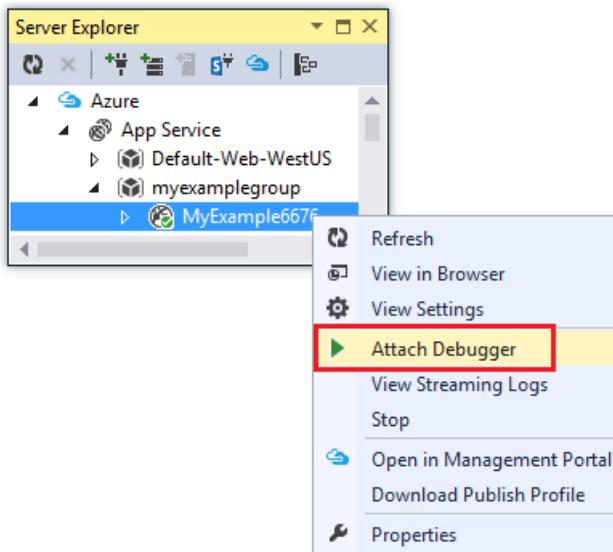
1. Open the web project that you created in [Create an ASP.NET app in Azure App Service](#).
2. Open *Controllers\HomeController.cs*.
3. Delete the `About()` method and insert the following code in its place.

```
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;

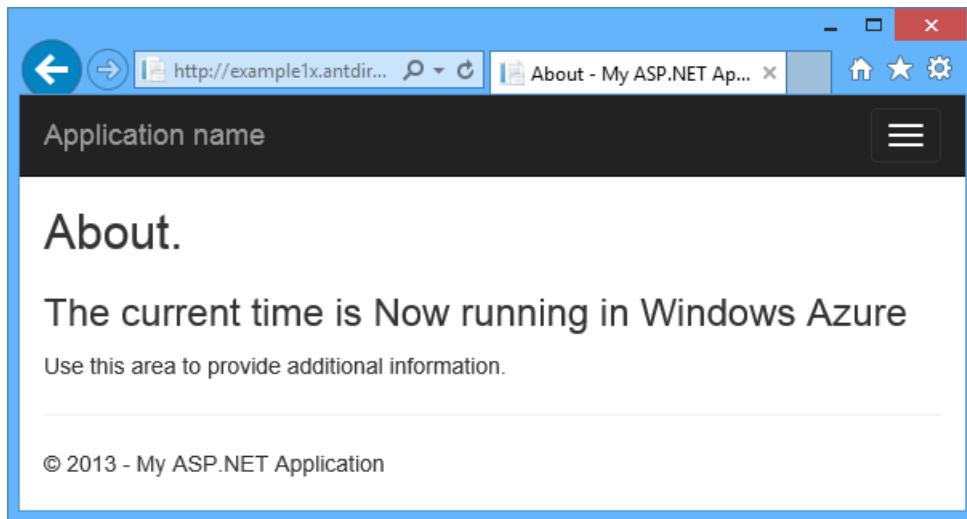
    return View();
}
```

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](#).

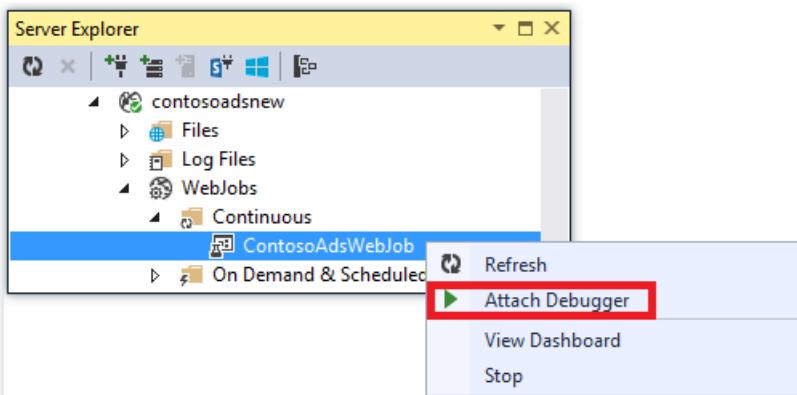
The features shown in this section are available only in Visual Studio 2013 with Update 4 or later.

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**.

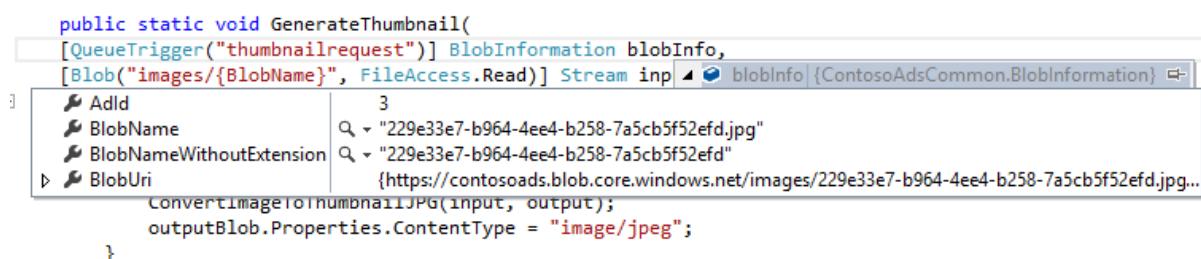


The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on an app in a 48-hour period. When you start debugging again in the same period, there 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.

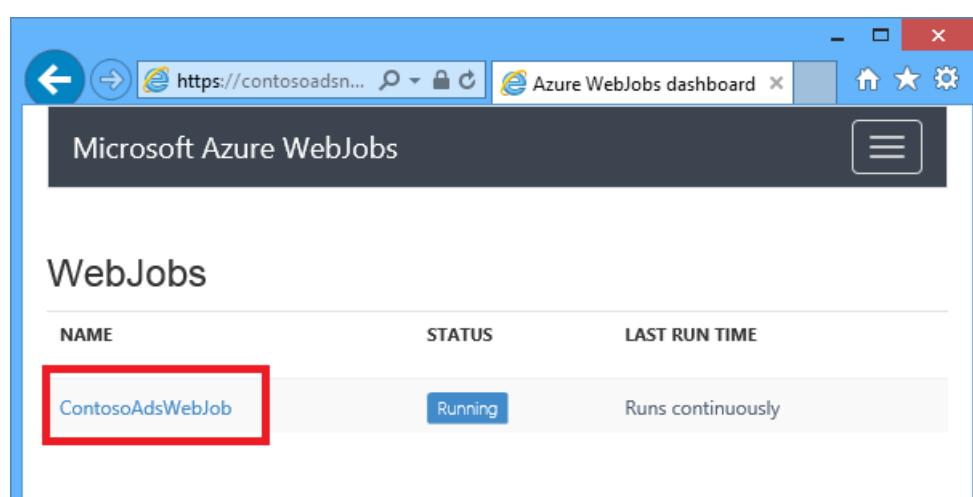
10. When the debugger breaks at your breakpoint, you can examine and change variable values while the program is running in the cloud. In the following illustration, the debugger shows the contents of the blobInfo object that was passed to the `GenerateThumbnail` method.



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. Below this, a green box indicates "Success 9 minutes ago (54 seconds running time)". A note says "⚡ New queue message detected on 'thumbnailrequest'." A table provides detailed information about the function execution:

PARAMETER	VALUE	NOTES
blobInfo	{"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}	
input	images/1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5.jpg	Read 116,652 bytes (100.15% of total). (about 596 milliseconds spent on I/O)
outputBlob	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();
}

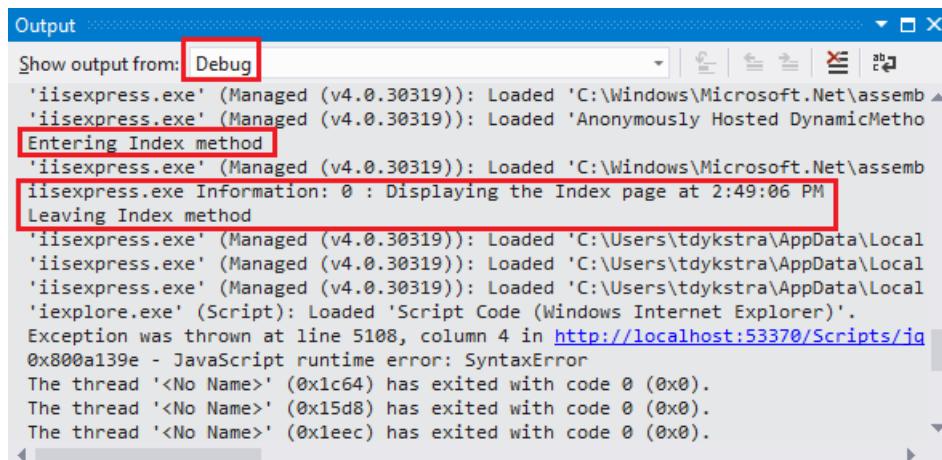
public ActionResult Contact()
{
    Trace.WriteLine("Entering Contact method");
    ViewBag.Message = "Your contact page.";
    Trace.LogError("Fatal error on the Contact page at " + DateTime.Now.ToString());
    Trace.WriteLine("Leaving Contact method");
    return View();
}
```

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 Output window in Visual Studio. The 'Show output from' dropdown is set to 'Debug'. The window displays several lines of trace output. The lines are highlighted with red boxes to show specific entries:

- '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 **Displaying the Index page at 2:49:06 PM**
- '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:

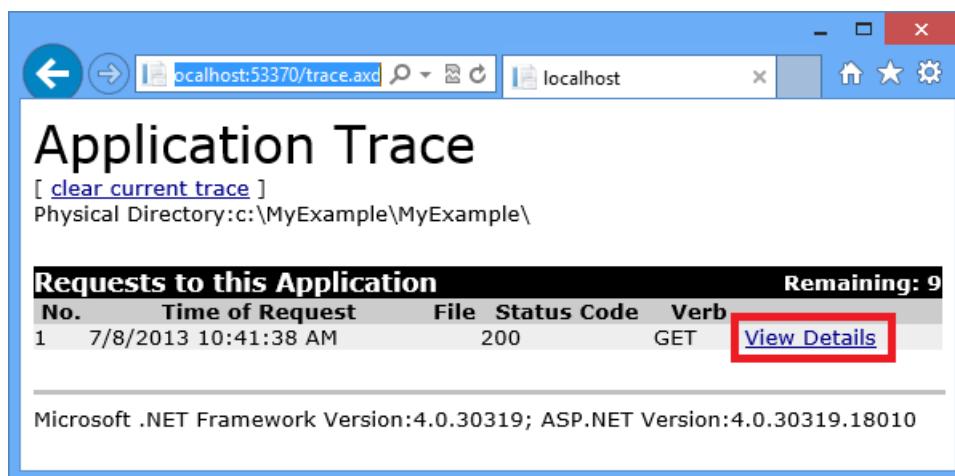
```
<system.diagnostics>
<trace>
  <listeners>
    <add name="WebPageTraceListener"
      type="System.Web.WebPageTraceListener,
      System.Web,
      Version=4.0.0.0,
      Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
  </listeners>
</trace>
</system.diagnostics>
```

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.

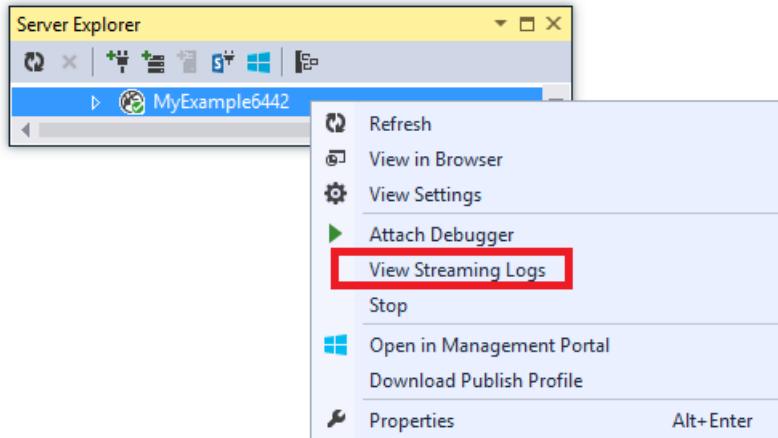
View the tracing output in Azure

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:

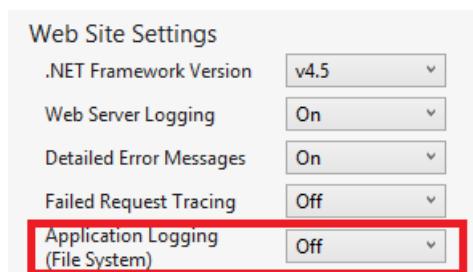
```

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.

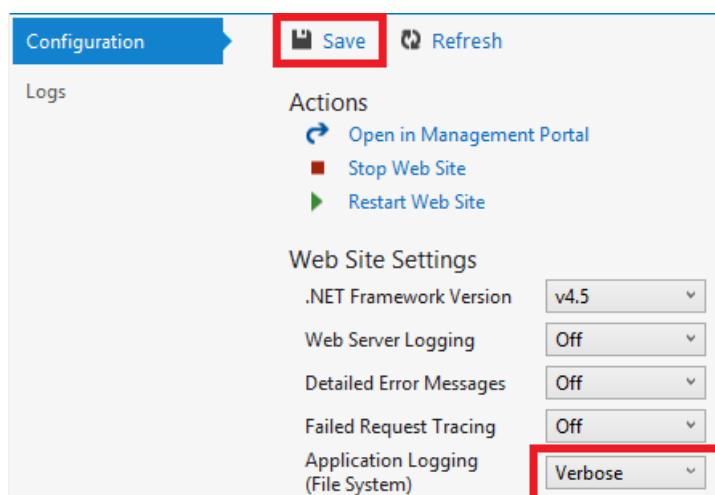
Visual Studio is only showing error-level traces because that is the default setting when you enable the log monitoring service. When you create a new App Service app, all logging is disabled by default, as you saw when you opened the settings page earlier:



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.

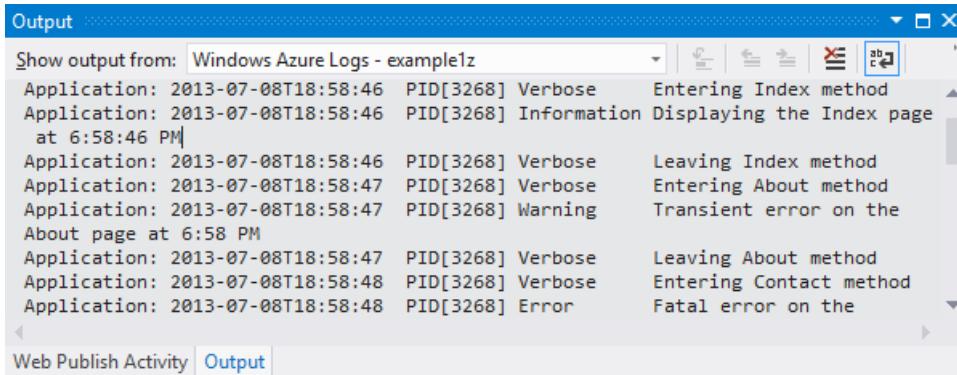
- Change **Application Logging (File System)** to **Verbose**, and then click **Save**.



- 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:

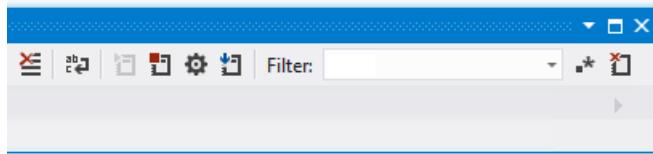
```
Application: 2013-07-08T18:58:46 PID[3268] Verbose Entering Index method
Application: 2013-07-08T18:58:46 PID[3268] Information Displaying the Index page
at 6:58:46 PM
Application: 2013-07-08T18:58:46 PID[3268] Verbose Leaving Index method
Application: 2013-07-08T18:58:47 PID[3268] Verbose Entering 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
```

Below the log entries, there are two tabs: "Web Publish Activity" and "Output". The "Output" tab is selected.

In this section, you enabled and disabled logging by using app settings. You can also enable and disable trace listeners by modifying the Web.config file. However, modifying the Web.config file causes the app domain to recycle, while enabling logging via the app configuration doesn't do that. If the problem takes a long time to reproduce, or is intermittent, recycling the app domain might "fix" it and force you to wait until it happens again. Enabling diagnostics in Azure lets you start capturing error information immediately without recycling the app domain.

Output window features

The **Microsoft Azure Logs** tab of the **Output** Window has several buttons and a text box:



These perform the following functions:

- Clear the **Output** window.
- Enable or disable word wrap.
- Start or stop monitoring logs.
- Specify which logs to monitor.
- Download logs.
- Filter logs based on a search string or a regular expression.
- Close the **Output** window.

If you enter a search string or regular expression, Visual Studio filters logging information at the client. That means you can enter the criteria after the logs are displayed in the **Output** window and you can change filtering criteria without having to regenerate the logs.

View web server logs

Web server logs record all HTTP activity for the app. In order to see them in the **Output** window, you must enable them for the app and tell Visual Studio that you want to monitor them.

1. In the **Azure Web App Configuration** tab that you opened from **Server Explorer**, change Web Server Logging to **On**, and then click **Save**.

Configuration

Save Refresh

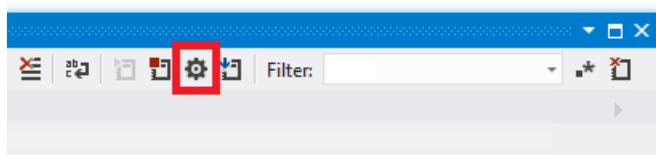
Logs Actions

- Open in Management Portal
- Stop Web Site
- Restart Web Site

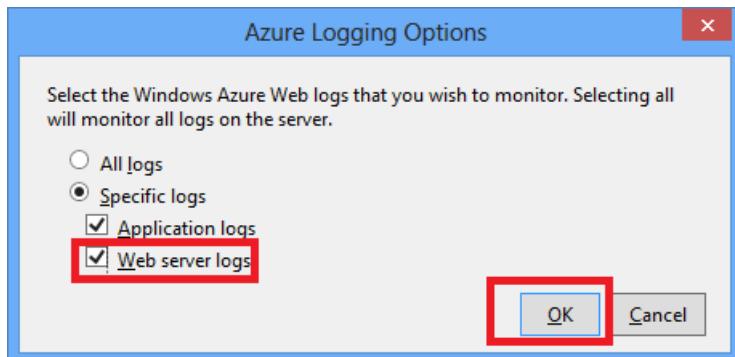
Web Site Settings

.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Verbose

2. In the **Output** Window, click the **Specify which Microsoft Azure logs to monitor** button.



3. In the **Microsoft Azure Logging Options** dialog box, select **Web server logs**, and then click **OK**.



4. In the browser window that shows the app, click **Home**, then click **About**, and then click **Contact**.

The application logs generally appear first, followed by the web server logs. You might have to wait a while for the logs to appear.

Output

Show output from: Windows Azure Logs - example1z

```

Connecting to Application logs ...
2013-07-08T19:50:34 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:51:34 No new trace in the past 1 min(s).
Application: 2013-07-08T19:52:34 No new trace in the past 2 min(s).
Connecting to Web server logs ...
2013-07-08T19:52:36 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:53:34 No new trace in the past 3 min(s).
Web server: 2013-07-08T19:53:36 No new trace in the past 1 min(s).
Application: 2013-07-08T19:54:34 No new trace in the past 4 min(s).
Web server: 2013-07-08T19:54:36 No new trace in the past 2 min(s).
Application: 2013-07-08T19:55:06 PID[3268] Information DotNetOpenAuth.Core,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=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

Logs

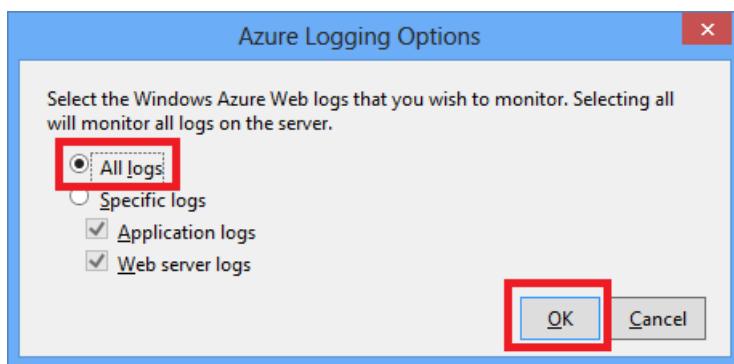
Actions

- Open in Management Portal
- Stop Web Site
- Restart Web Site

Web Site Settings

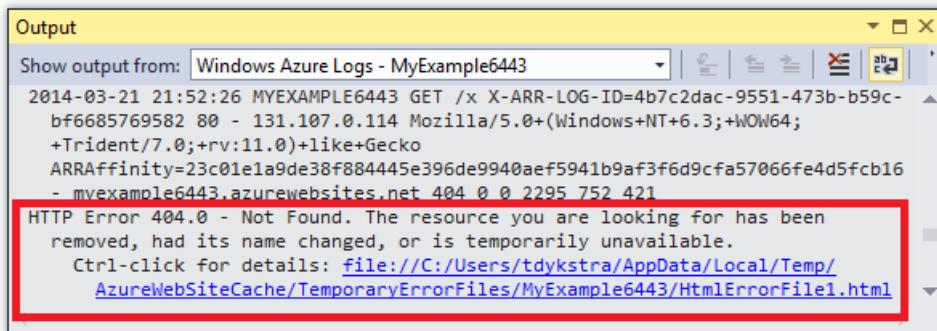
.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	On
Failed Request Tracing	Off
Application Logging (File System)	Verbose

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.



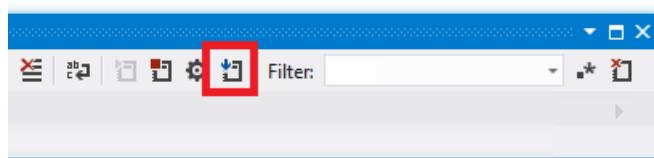
Control+click the link to see the log output formatted in a browser:

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." Below this is a section titled "Most likely causes:" with three bullet points: "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: "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. The "More Information:" section states that the error means the file or directory does not exist on the server and suggests creating it or trying again. It also links to "View more information >" and Microsoft Knowledge Base Articles.

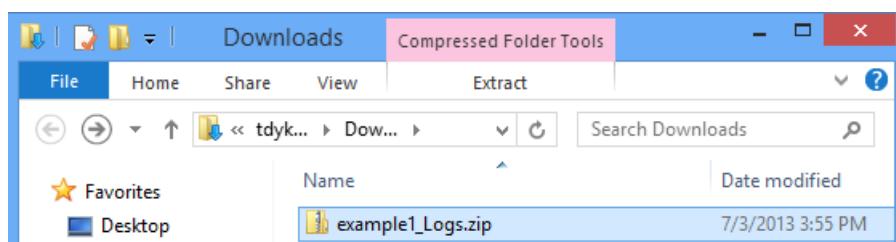
Download file system logs

Any logs that you can monitor in the **Output** window can also be downloaded as a .zip file.

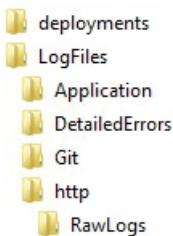
1. In the **Output** window, click **Download Streaming Logs**.



File Explorer opens to your *Downloads* folder with the downloaded file selected.



2. Extract the .zip file, and you see the following folder structure:



- Application tracing logs are in .txt files in the *LogFiles\Application* folder.
- Web server logs are in .log files in the *LogFiles\http\RawLogs* folder. You can use a tool such as [Log Parser](#) to view and manipulate these files.
- Detailed error message logs are in .html files in the *LogFiles\DetailedErrors* folder.

(The *deployments* folder is for files created by source control publishing; it doesn't have anything related to Visual Studio publishing. The *Git* folder is for traces related to source control publishing and the log file streaming service.)

View failed request tracing logs

Failed request tracing logs are useful when you need to understand the details of how IIS is handling an HTTP request, in scenarios such as URL rewriting or authentication problems.

App Service apps use the same failed request tracing functionality that has been available with IIS 7.0 and later. You don't have access to the IIS settings that configure which errors get logged, however. When you enable failed request tracing, all errors are captured.

You can enable failed request tracing by using Visual Studio, but you can't view them in Visual Studio. These logs are XML files. The streaming log service only monitors files that are deemed readable in plain text mode: .txt, .html, and .log files.

You can view failed request tracing logs in a browser directly via FTP or locally after using an FTP tool to download them to your local computer. In this section, you'll view them in a browser directly.

1. In the **Configuration** tab of the **Azure Web App** window that you opened from **Server Explorer**, change **Failed Request Tracing** to **On**, and then click **Save**.

A screenshot of the Azure Web App configuration interface. The 'Configuration' tab is selected. At the top right are 'Save' and 'Refresh' buttons. Below is a 'Actions' section with 'Open in Management Portal', 'Stop Web Site', and 'Restart Web Site' options. Under 'Web Site Settings', there are dropdown menus for '.NET Framework Version' (set to v4.5), 'Web Server Logging' (set to On), 'Detailed Error Messages' (set to On), and 'Failed Request Tracing' (which is highlighted with a red box and set to On). There is also an 'Application Logging (File System)' setting (set to 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

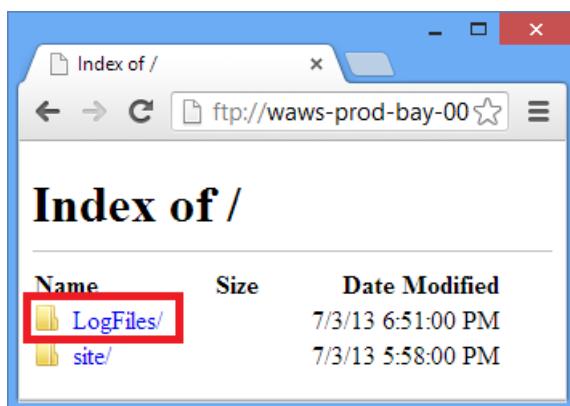
NOTE

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".

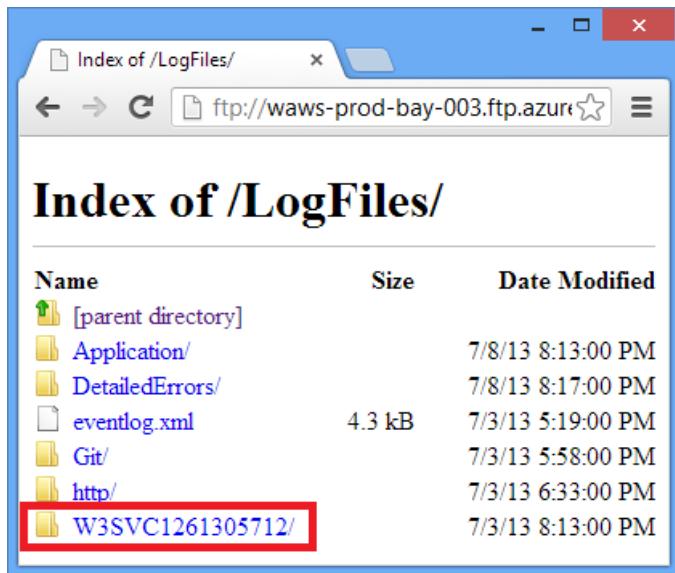
5. In a new browser window, go to the URL that is shown under **FTP hostname** or **FTPS hostname** in the **Overview** page for your app.
6. Sign in using the FTP credentials that you created earlier (including the app name prefix for the user name).

The browser shows the root folder of the app.

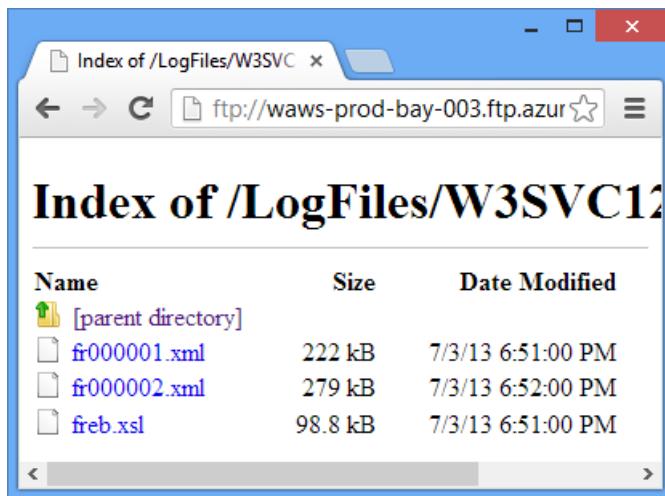
7. Open the *LogFiles* folder.



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 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 titled "Kudu Remote Execution Console". The console output shows the Windows command prompt environment, with the path D:\home\site\wwwroot. The user has run the command "npm install v8-profiler", which is visible at the bottom of the console window.

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

```

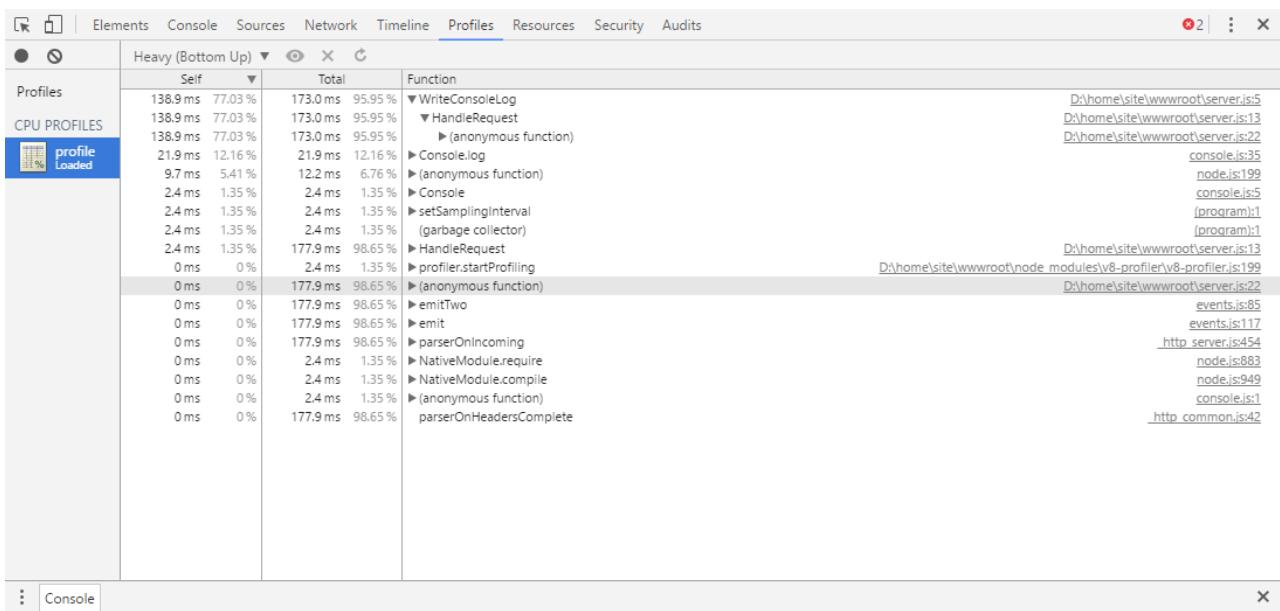
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new CMD process.
Type 'cls' to clear the console

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](#).

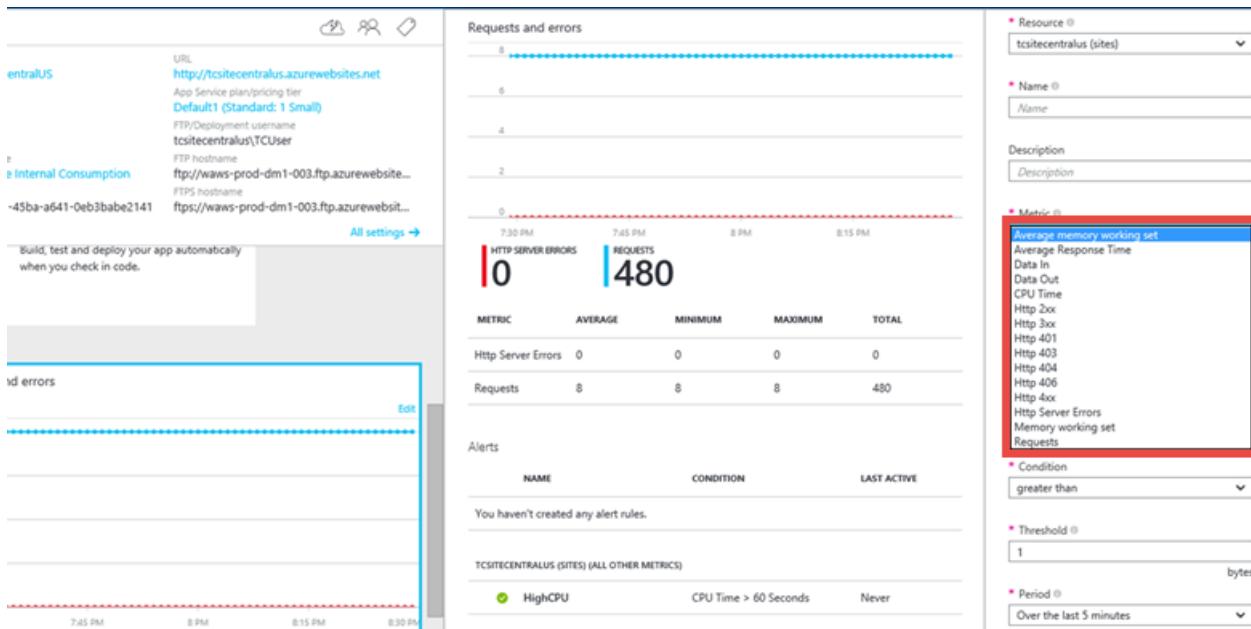
Monitor your app

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

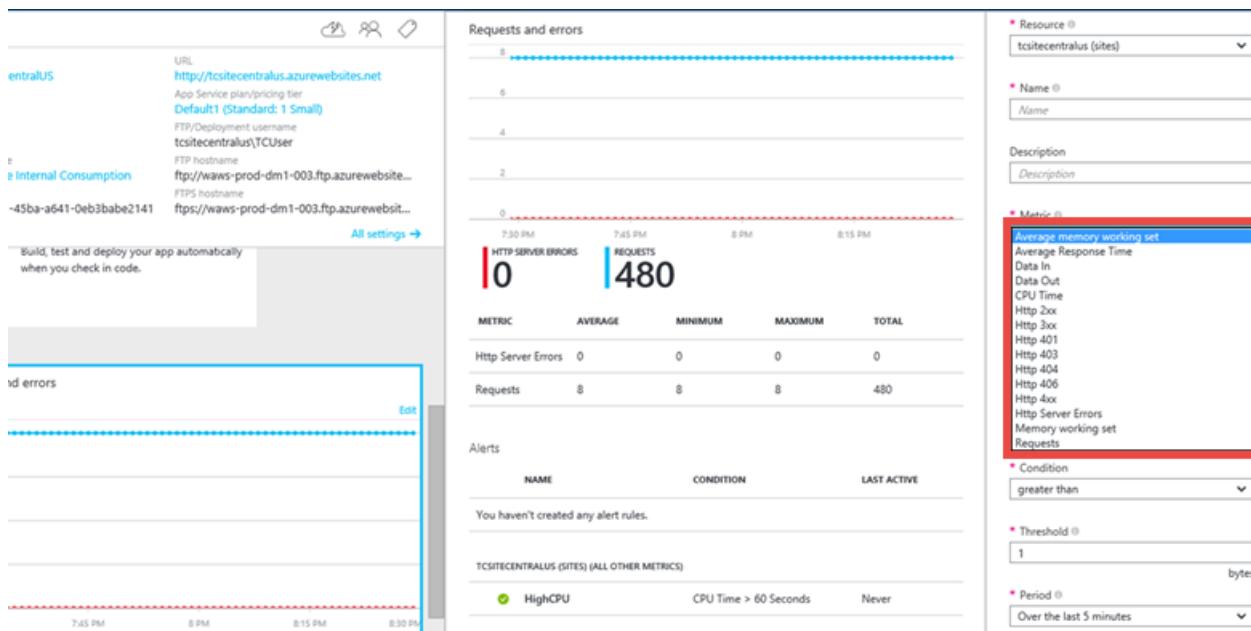
Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the [Azure portal](#). For more information, see [Track service health](#).

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](#).

Troubleshoot domain and SSL certificate problems in Azure App Service

1/14/2020 • 13 minutes to read • [Edit Online](#)

This article lists common problems that you might encounter when you configure a domain or SSL certificate for your web apps in Azure App Service. It also describes possible causes and solutions for these problems.

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN and Stack Overflow forums](#). Alternatively, you can file an Azure support incident. Go to the [Azure Support site](#) and select **Get Support**.

NOTE

This article has been updated to use the 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](#).

Certificate problems

You can't add an SSL certificate binding to an app

Symptom

When you add an SSL binding, you receive the following error message:

"Failed to add SSL binding. Cannot set certificate for existing VIP because another VIP already uses that certificate."

Cause

This problem can occur if you have multiple IP-based SSL bindings for the same IP address across multiple apps. For example, app A has an IP-based SSL with an old certificate. App B has an IP-based SSL with a new certificate for the same IP address. When you update the app SSL binding with the new certificate, it fails with this error because the same IP address is being used for another app.

Solution

To fix this problem, use one of the following methods:

- Delete the IP-based SSL binding on the app that uses the old certificate.
- Create a new IP-based SSL binding that uses the new certificate.

You can't delete a certificate

Symptom

When you try to delete a certificate, you receive the following error message:

"Unable to delete the certificate because it is currently being used in an SSL binding. The SSL binding must be removed before you can delete the certificate."

Cause

This problem might occur if another app uses the certificate.

Solution

Remove the SSL binding for that certificate from the apps. Then try to delete the certificate. If you still can't delete the certificate, clear the internet browser cache and reopen the Azure portal in a new browser window. Then try to

delete the certificate.

You can't purchase an App Service certificate

Symptom

You can't purchase an [Azure App Service certificate](#) from the Azure portal.

Cause and solution

This problem can occur for any of the following reasons:

- The App Service plan is Free or Shared. These pricing tiers don't support SSL.

Solution: Upgrade the App Service plan for app to Standard.

- The subscription doesn't have a valid credit card.

Solution: Add a valid credit card to your subscription.

- The subscription offer doesn't support purchasing an App Service certificate such as Microsoft Student.

Solution: Upgrade your subscription.

- The subscription reached the limit of purchases that are allowed on a subscription.

Solution: App Service certificates have a limit of 10 certificate purchases for the Pay-As-You-Go and EA subscription types. For other subscription types, the limit is 3. To increase the limit, contact [Azure support](#).

- The App Service certificate was marked as fraud. You received the following error message: "Your certificate has been flagged for possible fraud. The request is currently under review. If the certificate does not become usable within 24 hours, contact Azure Support."

Solution: If the certificate is marked as fraud and isn't resolved after 24 hours, follow these steps:

1. Sign in to the [Azure portal](#).
2. Go to **App Service Certificates**, and select the certificate.
3. Select **Certificate Configuration > Step 2: Verify > Domain Verification**. This step sends an email notice to the Azure certificate provider to resolve the problem.

Custom domain problems

A custom domain returns a 404 error

Symptom

When you browse to the site by using the custom domain name, you receive the following error message:

"Error 404-Web app not found."

Cause and solution

Cause 1

The custom domain that you configured is missing a CNAME or A record.

Solution for cause 1

- If you added an A record, make sure that a TXT record is also added. For more information, see [Create the A record](#).
- If you don't have to use the root domain for your app, we recommend that you use a CNAME record instead of an A record.
- Don't use both a CNAME record and an A record for the same domain. This issue can cause a conflict and prevent the domain from being resolved.

Cause 2

The internet browser might still be caching the old IP address for your domain.

Solution for Cause 2

Clear the browser. For Windows devices, you can run the command `ipconfig /flushdns`. Use [WhatsmyDNS.net](#) to verify that your domain points to the app's IP address.

You can't add a subdomain

Symptom

You can't add a new host name to an app to assign a subdomain.

Solution

- Check with subscription administrator to make sure that you have permissions to add a host name to the app.
- If you need more subdomains, we recommend that you change the domain hosting to Azure Domain Name Service (DNS). By using Azure DNS, you can add 500 host names to your app. For more information, see [Add a subdomain](#).

DNS can't be resolved

Symptom

You received the following error message:

"The DNS record could not be located."

Cause

This problem occurs for one of the following reasons:

- The time to live (TTL) period has not expired. Check the DNS configuration for your domain to determine the TTL value, and then wait for the period to expire.
- The DNS configuration is incorrect.

Solution

- Wait for 48 hours for this problem to resolve itself.
- If you can change the TTL setting in your DNS configuration, change the value to 5 minutes to see whether this resolves the problem.
- Use [WhatsmyDNS.net](#) to verify that your domain points to the app's IP address. If it doesn't, configure the A record to the correct IP address of the app.

You need to restore a deleted domain

Symptom

Your domain is no longer visible in the Azure portal.

Cause

The owner of the subscription might have accidentally deleted the domain.

Solution

If your domain was deleted fewer than seven days ago, the domain has not yet started the deletion process. In this case, you can buy the same domain again on the Azure portal under the same subscription. (Be sure to type the exact domain name in the search box.) You won't be charged again for this domain. If the domain was deleted more than seven days ago, contact [Azure support](#) for help with restoring the domain.

Domain problems

You purchased an SSL certificate for the wrong domain

Symptom

You purchased an App Service certificate for the wrong domain. You can't update the certificate to use the correct domain.

Solution

Delete that certificate and then buy a new certificate.

If the current certificate that uses the wrong domain is in the "Issued" state, you'll also be billed for that certificate. App Service certificates are not refundable, but you can contact [Azure support](#) to see whether there are other options.

An App Service certificate was renewed, but the app shows the old certificate

Symptom

The App Service certificate was renewed, but the app that uses the App Service certificate is still using the old certificate. Also, you received a warning that the HTTPS protocol is required.

Cause

App Service automatically syncs your certificate within 48 hours. When you rotate or update a certificate, sometimes the application is still retrieving the old certificate and not the newly updated certificate. The reason is that the job to sync the certificate resource hasn't run yet. Click Sync. The sync operation automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

Solution

You can force a sync of the certificate:

1. Sign in to the [Azure portal](#). Select **App Service Certificates**, and then select the certificate.
2. Select **Rekey and Sync**, and then select **Sync**. The sync takes some time to finish.
3. When the sync is completed, you see the following notification: "Successfully updated all the resources with the latest certificate."

Domain verification is not working

Symptom

The App Service certificate requires domain verification before the certificate is ready to use. When you select **Verify**, the process fails.

Solution

Manually verify your domain by adding a TXT record:

1. Go to the Domain Name Service (DNS) provider that hosts your domain name.
2. Add a TXT record for your domain that uses the value of the domain token that's shown in the Azure portal.

Wait a few minutes for DNS propagation to run, and then select the **Refresh** button to trigger the verification.

As an alternative, you can use the HTML webpage method to manually verify your domain. This method allows the certificate authority to confirm the domain ownership of the domain that the certificate is issued for.

1. Create an HTML file that's named {domain verification token}.html. The content of this file should be the value of domain verification token.
2. Upload this file at the root of the web server that's hosting your domain.
3. Select **Refresh** to check the certificate status. It might take few minutes for verification to finish.

For example, if you're buying a standard certificate for azure.com with the domain verification token 1234abcd, a web request made to <https://azure.com/1234abcd.html> should return 1234abcd.

IMPORTANT

A certificate order has only 15 days to complete the domain verification operation. After 15 days, the certificate authority denies the certificate, and you are not charged for the certificate. In this situation, delete this certificate and try again.

You can't purchase a domain

Symptom

You can't buy an App Service domain in the Azure portal.

Cause and solution

This problem occurs for one of the following reasons:

- There's no credit card on the Azure subscription, or the credit card is invalid.

Solution: Add a valid credit card to your subscription.

- You're not the subscription owner, so you don't have permission to purchase a domain.

Solution: Assign the Owner role to your account. Or contact the subscription administrator to get permission to purchase a domain.

- You have reached the limit for purchasing domains on your subscription. The current limit is 20.

Solution: To request an increase to the limit, contact [Azure support](#).

- Your Azure subscription type does not support the purchase of an App Service domain.

Solution: Upgrade your Azure subscription to another subscription type, such as a Pay-As-You-Go subscription.

You can't add a host name to an app

Symptom

When you add a host name, the process fails to validate and verify the domain.

Cause

This problem occurs for one of the following reasons:

- You don't have permission to add a host name.

Solution: Ask the subscription administrator to give you permission to add a host name.

- Your domain ownership could not be verified.

Solution: Verify that your CNAME or A record is configured correctly. To map a custom domain to an app, create either a CNAME record or an A record. If you want to use a root domain, you must use A and TXT records:

RECORD TYPE	HOST	POINT TO
A	@	IP address for an app
TXT	@	<app-name>.azurewebsites.net
CNAME	www	<app-name>.azurewebsites.net

FAQ

Do I have to configure my custom domain for my website once I buy it?

When you purchase a domain from the Azure portal, the App Service application is automatically configured to use that custom domain. You don't have to take any additional steps. For more information, watch [Azure App Service Self Help: Add a Custom Domain Name](#) on Channel9.

Can I use a domain purchased in the Azure portal to point to an Azure VM instead?

Yes, you can point the domain to a VM. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

Is my domain hosted by GoDaddy or Azure DNS?

App Service Domains use GoDaddy for domain registration and Azure DNS to host the domains.

I have auto-renew enabled but still received a renewal notice for my domain via email. What should I do?

If you have auto-renew enabled, you do not need to take any action. The notice email is provided to inform you that the domain is close to expiring and to renew manually if auto-renew is not enabled.

Will I be charged for Azure DNS hosting my domain?

The initial cost of domain purchase applies to domain registration only. In addition to the registration cost, there are incurring charges for Azure DNS based on your usage. For more information, see [Azure DNS pricing](#) for more details.

I purchased my domain earlier from the Azure portal and want to move from GoDaddy hosting to Azure DNS hosting. How can I do this?

It is not mandatory to migrate to Azure DNS hosting. If you do want to migrate to Azure DNS, the domain management experience in the Azure portal about provides information on steps necessary to move to Azure DNS. If the domain was purchased through App Service, migration from GoDaddy hosting to Azure DNS is relatively seamless procedure.

I would like to purchase my domain from App Service Domain but can I host my domain on GoDaddy instead of Azure DNS?

Beginning July 24, 2017, App Service domains purchased in the portal are hosted on Azure DNS. If you prefer to use a different hosting provider, you must go to their website to obtain a domain hosting solution.

Do I have to pay for privacy protection for my domain?

When you purchase a domain through the Azure portal, you can choose to add privacy at no additional cost. This is one of the benefits of purchasing your domain through Azure App Service.

If I decide I no longer want my domain, can I get my money back?

When you purchase a domain, you are not charged for a period of five days, during which time you can decide that you do not want the domain. If you do decide you don't want the domain within that five-day period, you are not charged. (.uk domains are an exception to this. If you purchase a .uk domain, you are charged immediately and you cannot be refunded.)

Can I use the domain in another Azure App Service app in my subscription?

Yes. When you access the Custom Domains and SSL blade in the Azure portal, you see the domains that you have purchased. You can configure your app to use any of those domains.

Can I transfer a domain from one subscription to another subscription?

You can move a domain to another subscription/resource group using the [Move-AzResource](#) PowerShell cmdlet.

How can I manage my custom domain if I don't currently have an Azure App Service app?

You can manage your domain even if you don't have an App Service Web App. Domain can be used for Azure services like Virtual machine, Storage etc. If you intend to use the domain for App Service Web Apps, then you need to include a Web App that is not on the Free App Service plan in order to bind the domain to your web app.

Can I move a web app with a custom domain to another subscription or from App Service Environment v1 to V2?

Yes, you can move your web app across subscriptions. Follow the guidance in [How to move resources in Azure](#).

There are a few limitations when moving the web app. For more information, see [Limitations for moving App Service resources](#).

After moving the web app, the host name bindings of the domains within the custom domains setting should remain the same. No additional steps are required to configure the host name bindings.

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](#).

How to prepare for an inbound IP address change

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

If you received a notification that the inbound IP address of your Azure App Service app is changing, follow the instructions in this article.

Determine if you have to do anything

- Option 1: If your App Service app does not have a Custom Domain, no action is required.
- Option 2: If only a CNAME record (DNS record pointing to a URI) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), no action is required.
- Option 3: If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the new one. You can find the new IP address by following the instructions in the next section.
- Option 4: If your application is behind a load balancer, IP Filter, or any other IP mechanism that requires your app's IP address, replace the existing IP address with the new one. You can find the new IP address by following the instructions in the next section.

Find the new inbound IP Address in the Azure portal

The new inbound IP address that is being given to your app is in the portal in the **Virtual IP address** field. Both this new IP address and the old one are connected to your app now, and later the old one will be disconnected.

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. If the app is a function app, see [Function app inbound IP address](#).
5. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.
6. Copy the IP address and reconfigure your domain record or IP mechanism.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

How to prepare for an outbound IP address change

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

If you received a notification that the outbound IP addresses of your Azure App Service app are changing, follow the instructions in this article.

Determine if you have to do anything

- Option 1: If your App Service app does not use IP filtering, an explicit inclusion list, or special handling of outbound traffic such as routing or firewall, no action is required.
- Option 2: If your app does have special handling for the outbound IP addresses (see examples below), add the new outbound IP addresses wherever the existing ones appear. Don't replace the existing IP addresses. You can find the new outbound IP addresses by following the instructions in the next section.

For example, an outbound IP address may be explicitly included in a firewall outside your app, or an external payment service may have an allowed list that contains the outbound IP address for your app. If your outbound address is configured in a list anywhere outside your app, that needs to change.

Find the outbound IP addresses in the Azure portal

The new outbound IP addresses are shown in the portal before they take effect. When Azure starts using the new ones, the old ones will no longer be used. Only one set at a time is used, so entries in inclusion lists must have both old and new IP addresses to prevent an outage when the switch happens.

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. If the app is a function app, see [Function app outbound IP addresses](#).
5. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Outbound IP addresses**.
6. Copy the IP addresses, and add them to your special handling of outbound traffic such as a filter or allowed list. Don't delete the existing IP addresses in the list.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

How to prepare for an SSL IP address change

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

If you received a notification that the SSL IP address of your Azure App Service app is changing, follow the instructions in this article to release existing SSL IP address and assign a new one.

Release SSL IP addresses and assign new ones

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **SSL settings** in the left navigation.
5. In the SSL bindings section, select the host name record. In the editor that opens, choose **SNI SSL** on the **SSL Type** drop-down menu and click **Add Binding**. When you see the operation success message, the existing IP address has been released.
6. In the **SSL bindings** section, again select the same host name record with the certificate. In the editor that opens, this time choose **IP Based SSL** on the **SSL Type** drop-down menu and click **Add Binding**. When you see the operation success message, you've acquired a new IP address.
7. If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the newly generated one. You can find the new IP address by following the instructions in the next section.

Find the new SSL IP address in the Azure Portal

1. Wait a few minutes, and then open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.
5. Copy the IP address and reconfigure your domain record or IP mechanism.

Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [SSL and SSL IP addresses in Azure App Service](#).