

Table of Contents

[App Service Documentation](#)

[Overview](#)

[About Web Apps](#)

[About Web App for Containers](#)

[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 Python app](#)

[Create static HTML site](#)

[Tutorials](#)

[1 - App with DB](#)

[.NET with SQL DB](#)

[PHP with MySQL](#)

[Node.js with MongoDB](#)

[Java with MySQL](#)

[1 - Create RESTful API](#)

[2 - Map Custom Domain](#)

[3 - Bind SSL Certificate](#)

[4 - Add CDN](#)

[Samples](#)

[Azure CLI](#)

[Azure PowerShell](#)

[Concepts](#)

[How App Service works](#)

- [App Service plans](#)
- [OS functionality](#)
- [Authentication and authorization](#)
- [Hybrid connections](#)
- [Traffic Manager integration](#)
- [Local cache](#)
- [Diagnostics](#)
- [How-To guides](#)
 - [Configure app](#)
 - [Use app settings](#)
 - [Upload existing Java app](#)
 - [Configure PHP](#)
 - [Python](#)
 - [Connect to on-prem resources](#)
 - [Deploy to Azure](#)
 - [Deploy the app](#)
 - [Set deployment credentials](#)
 - [Create staging environments](#)
 - [Map custom domain](#)
 - [Buy domain](#)
 - [Map domains with Traffic Manager](#)
 - [Migrate an active domain](#)
 - [Secure app](#)
 - [Buy SSL cert](#)
 - [Authenticate users](#)
 - [Use Managed Service Identity](#)
 - [Use SSL cert in application code](#)
 - [Configure TLS mutual authentication](#)
 - [Scale app](#)
 - [Scale up server capacity](#)
 - [Configure PremiumV2 tier](#)
 - [Scale out to multiple instances](#)

[Monitor app](#)

[Quotas & alerts](#)

[Enable logs](#)

[Manage app](#)

[Manage the hosting plan](#)

[Back up an app](#)

[Restore a backup](#)

[Clone an app](#)

[Move resources](#)

[Run background tasks](#)

[Create WebJobs](#)

[Develop WebJobs using VS](#)

[Reference](#)

[Azure CLI](#)

[Azure PowerShell](#)

[REST API](#)

[Resources](#)

[Azure Roadmap](#)

[Pricing](#)

[Quota Information](#)

[Service Updates & Release Notes](#)

[Azure Updates](#)

[Azure SDK for .NET 3.0](#)

[Azure SDK for .NET 2.9](#)

[Azure SDK for .NET 2.8](#)

[Azure SDK for .NET 2.7](#)

[Azure SDK for .NET 2.6](#)

[Azure SDK for .NET 2.5.1](#)

[Azure SDK for .NET 2.5](#)

[Azure SDK for .NET 2.4](#)

[Azure SDK for .NET 2.3](#)

[Azure SDK for .NET 2.2](#)

[Azure SDK for .NET 2.1](#)

[Azure SDK for .NET 2.0](#)

[Best practices](#)

[Samples](#)

[Videos](#)

[Cookbooks](#)

[Reference Architectures](#)

[Deployment Scripts](#)

[Troubleshooting](#)

[Troubleshoot with Visual Studio](#)

[Troubleshoot Node.js app](#)

[Troubleshoot HTTP 502 & 503](#)

[Troubleshoot performance issues](#)

[FAQ](#)

Web Apps overview

10/30/2017 • 2 min to read • [Edit Online](#)

Azure App Service Web Apps (or just Web Apps) is a 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. You can run and scale apps with ease on Windows or Linux VMs (see [App Service on Linux](#)).

Web Apps 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 VSTS, 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 Web Apps on. For more information, see [App Service plans in Azure Web Apps](#).

The following 5-minute video introduces Azure App Service Web Apps.

Why use Web Apps?

Here are some key features of App Service Web Apps:

- **Multiple languages and frameworks** - Web Apps 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 Visual Studio Team Services, GitHub, BitBucket, Docker Hub, or Azure Container Service. Promote updates through [test and staging environments](#). Manage your apps in Web Apps 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** - Web Apps 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 Web Apps in App Service, Azure offers other services that can be used for hosting websites and web applications. For most scenarios, Web Apps 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](#)

[PHP](#)

[Node.js](#)

[Java](#)

[Python](#)

[HTML](#)

Introduction to Azure App Service on Linux

10/24/2017 • 2 min to read • [Edit Online](#)

[Web App](#) 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. The following sections lists the application stacks that are currently supported.

Languages

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

LANGUAGE	SUPPORTED VERSIONS
Node.js	4.4, 4.5, 6.2, 6.6, 6.9-6.11, 8.0, 8.1
PHP	5.6, 7.0
.NET Core	1.0, 1.1
Ruby	2.3

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.

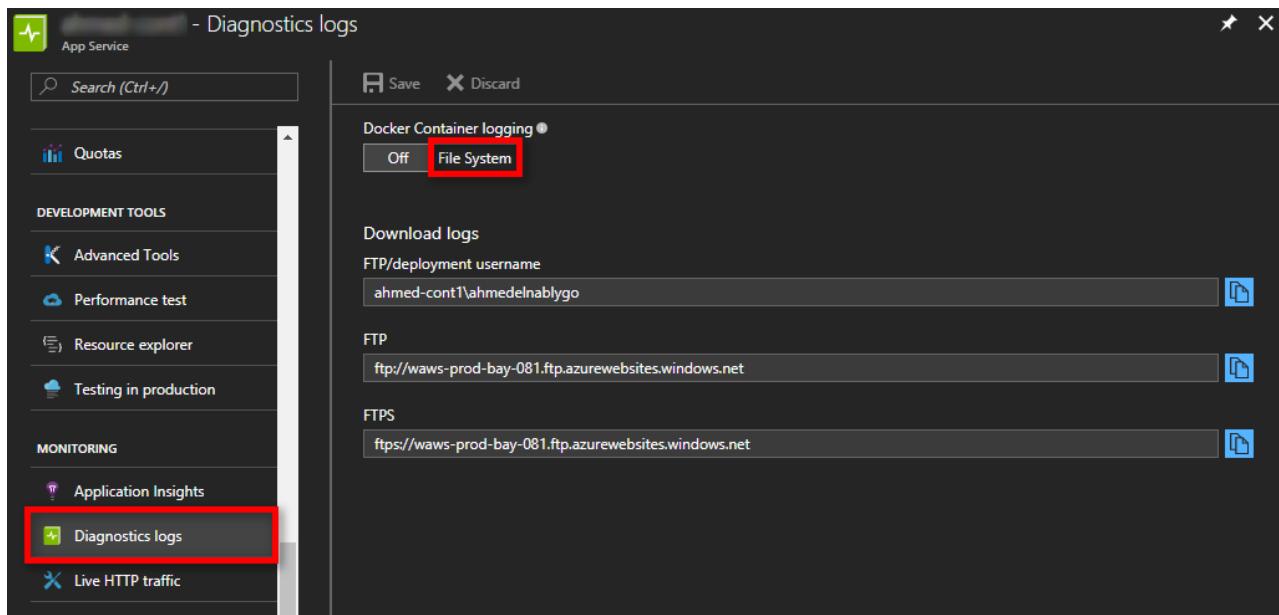
Some features, such as virtual network integration, Azure Active Directory/third-party authentication, or Kudu site extensions, are not available yet. Once these features are available, we will update our documentation and blog about the changes.

App Service on Linux is only supported with [Basic and Standard](#) app service plans and does not have a [Free or Shared](#) tier. The following are also important restrictions for App Service on Linux:

- You cannot create Web App for Containers in an App Service plan already hosting non-Linux Web Apps.
- When creating Web App for Containers in a resource group containing non-Linux Web Apps, you must create an App Service plan in a different region than the existing App Service plan.

Troubleshooting

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 **Docker Container logging** under **Diagnostics Logs**.



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

RD0003FF5A6A01:
docker.log

RD0003FFA76731:
docker.log

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

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

Next steps

See the following links to get started with App Service on Linux. You can post questions and concerns on [our forum](#).

- [How to use a custom Docker image for Web App for Containers](#)
- [Using .NET Core in Azure App Service on Linux](#)
- [Using Ruby in Azure App Service on Linux](#)
- [Azure App Service Web App for Containers FAQ](#)
- [SSH support for Azure App Service on Linux](#)
- [Set up staging environments in Azure App Service](#)
- [Docker Hub Continuous Deployment with Web App for Containers](#)

Introduction to App Service environments

12/4/2017 • 4 min to read • [Edit Online](#)

Overview

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 web apps, [mobile apps](#), API apps, and [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.

- ASEs enable high-scale app hosting with secure network access. For more information, see the [AzureCon Deep Dive on ASEs](#).
- 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 [Configure a WAF for App Service environments](#).

Dedicated environment

An ASE is dedicated exclusively to a single subscription and can host 100 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

An ASE can be created only in an 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](#).

Azure App Service, Virtual Machines, Service Fabric, and Cloud Services comparison

1/2/2018 • 12 min to read • [Edit Online](#)

Overview

Azure offers several ways to host web sites: [Azure App Service](#), [Virtual Machines](#), [Service Fabric](#), and [Cloud Services](#). This article helps you understand the options and make the right choice for your web application.

Azure App Service is the best choice for most web apps. Deployment and management are integrated into the platform, sites can scale quickly to handle high traffic loads, and the built-in load balancing and traffic manager provide high availability. You can move existing sites to Azure App Service easily with an [online migration tool](#), use an open-source app from the Web Application Gallery, or create a new site using the framework and tools of your choice. The [WebJobs](#) feature makes it easy to add background job processing to your App Service web app.

Service Fabric is a good choice if you're creating a new app or re-writing an existing app to use a microservice architecture. Apps, which run on a shared pool of machines, can start small and grow to massive scale with hundreds or thousands of machines as needed. Stateful services make it easy to consistently and reliably store app state, and Service Fabric automatically manages service partitioning, scaling, and availability for you. Service Fabric also supports WebAPI with Open Web Interface for .NET (OWIN) and ASP.NET Core. Compared to App Service, Service Fabric also provides more control over, or direct access to, the underlying infrastructure. You can remote into your servers or configure server startup tasks. Cloud Services is similar to Service Fabric in degree of control versus ease of use, but it's now a legacy service and Service Fabric is recommended for new development.

If you have an existing application that would require substantial modifications to run in App Service or Service Fabric, you could choose Virtual Machines in order to simplify migrating to the cloud. However, correctly configuring, securing, and maintaining VMs requires much more time and IT expertise compared to Azure App Service and Service Fabric. If you are considering Azure Virtual Machines, make sure you take into account the ongoing maintenance effort required to patch, update, and manage your VM environment. Azure Virtual Machines is Infrastructure-as-a-Service (IaaS), while App Service and Service Fabric are Platform-as-a-Service (PaaS).

Feature Comparison

The following table compares the capabilities of App Service, Cloud Services, Virtual Machines, and Service Fabric to help you make the best choice. For current information about the SLA for each option, see [Azure Service Level Agreements](#).

FEATURE	APP SERVICE (WEB APPS)	CLOUD SERVICES (WEB ROLES)	VIRTUAL MACHINES	SERVICE FABRIC	NOTES
---------	------------------------	----------------------------	------------------	----------------	-------

FEATURE	APP SERVICE (WEB APPS)	CLOUD SERVICES (WEB ROLES)	VIRTUAL MACHINES	SERVICE FABRIC	NOTES
Near-instant deployment	X			X	Deploying an application or an application update to a Cloud Service, or creating a VM, takes several minutes at least; deploying an application to a web app takes seconds.
Scale up to larger machines without redeploy	X			X	
Web server instances share content and configuration, which means you don't have to redeploy or reconfigure as you scale.	X			X	
Multiple deployment environments (production and staging)	X	X		X	Service Fabric allows you to have multiple environments for your apps or to deploy different versions of your app side-by-side.
Automatic OS update management	X	X			Partially through Patch Orchestration Application (POA) and fully in the future.
Seamless platform switching (easily move between 32 bit and 64 bit)	X	X			
Deploy code with GIT, FTP	X		X		

FEATURE	APP SERVICE (WEB APPS)	CLOUD SERVICES (WEB ROLES)	VIRTUAL MACHINES	SERVICE FABRIC	NOTES
Deploy code with Web Deploy	X		X		<p>Cloud Services supports the use of Web Deploy to deploy updates to individual role instances. However, you can't use it for initial deployment of a role, and if you use Web Deploy for an update you have to deploy separately to each instance of a role. Multiple instances are required in order to qualify for the Cloud Service SLA for production environments.</p>
WebMatrix support	X		X		
Access to services like Service Bus, Storage, SQL Database	X	X	X	X	
Host web or web services tier of a multi-tier architecture	X	X	X	X	
Host middle tier of a multi-tier architecture	X	X	X	X	<p>App Service web apps can easily host a REST API middle tier, and the WebJobs feature can host background processing jobs. You can run WebJobs in a dedicated website to achieve independent scalability for the tier.</p>
Integrated MySQL-as-a-service support	X	X			

FEATURE	APP SERVICE (WEB APPS)	CLOUD SERVICES (WEB ROLES)	VIRTUAL MACHINES	SERVICE FABRIC	NOTES
Support for ASP.NET, classic ASP, Node.js, PHP, Python	X	X	X	X	Service Fabric supports the creation of a web front-end using ASP.NET 5 or you can deploy any type of application (Node.js, Java, etc) as a guest executable .
Scale out to multiple instances without redeploy	X	X	X	X	Virtual Machines can scale out to multiple instances, but the services running on them must be written to handle this scale-out. You have to configure a load balancer to route requests across the machines, and create an Affinity Group to prevent simultaneous restarts of all instances due to maintenance or hardware failures.
Support for SSL	X	X	X	X	For App Service web apps, SSL for custom domain names is only supported for Basic and Standard mode. For information about using SSL with web apps, see Configuring an SSL certificate for an Azure Website .
Visual Studio integration	X	X	X	X	
Remote Debugging	X	X	X		
Deploy code with TFS	X	X	X	X	

FEATURE	APP SERVICE (WEB APPS)	CLOUD SERVICES (WEB ROLES)	VIRTUAL MACHINES	SERVICE FABRIC	NOTES
Network isolation with Azure Virtual Network	X	X	X	X	See also Azure Websites Virtual Network Integration
Support for Azure Traffic Manager	X	X	X	X	
Integrated Endpoint Monitoring	X	X	X		
Remote desktop access to servers		X	X	X	
Install any custom MSI		X	X	X	Service Fabric allows you to host any executable file as a guest executable or you can install any app on the VMs.
Ability to define/execute start-up tasks		X	X	X	
Can listen to ETW events		X	X	X	

Scenarios and recommendations

Here are some common application scenarios with recommendations as to which Azure web hosting option might be most appropriate for each.

- [I need a web front end with background processing and database backend to run business applications integrated with on-premises assets.](#)
- [I need a reliable way to host my corporate website that scales well and offers global reach.](#)
- [I have an IIS6 application running on Windows Server 2003.](#)
- [I'm a small business owner, and I need an inexpensive way to host my site but with future growth in mind.](#)
- [I'm a web or graphic designer, and I want to design and build web sites for my customers.](#)
- [I'm migrating my multi-tier application with a web front-end to the Cloud.](#)
- [My application depends on highly customized Windows or Linux environments and I want to move it to the cloud.](#)
- [My site uses open source software, and I want to host it in Azure.](#)
- [I have a line-of-business application that needs to connect to the corporate network.](#)
- [I want to host a REST API or web service for mobile clients.](#)

I need a web front end with background processing and database backend to run business applications integrated with on-premises assets.

Azure App Service is a great solution for complex business applications. It lets you develop apps that scale automatically on a load balanced platform, are secured with Active Directory, and connect to your on-premises resources. It makes managing those apps easy through a world-class portal and APIs, and allows you to gain insight into how customers are using them with app insight tools. The [Webjobs](#) feature lets you run background processes and tasks as part of your web tier, while hybrid connectivity and VNET features make it easy to connect back to on-premises resources. Azure App Service provides three 9's SLA for web apps and enables you to:

- Run your applications reliably on a self-healing, auto-patching cloud platform.
- Scale automatically across a global network of datacenters.
- Back up and restore for disaster recovery.
- Be ISO, SOC2, and PCI compliant.
- Integrate with Active Directory

I need a reliable way to host my corporate website that scales well and offers global reach.

Azure App Service is a great solution for hosting corporate websites. It enables web apps to scale quickly and easily to meet demand across a global network of datacenters. It offers local reach, fault tolerance, and intelligent traffic management. All on a platform that provides world-class management tools, allowing you to gain insight into site health and site traffic quickly and easily. Azure App Service provides three 9's SLA for web apps and enables you to:

- Run your websites reliably on a self-healing, auto-patching cloud platform.
- Scale automatically across a global network of datacenters.
- Back up and restore for disaster recovery.
- Manage logs and traffic with integrated tools.
- Be ISO, SOC2, and PCI compliant.
- Integrate with Active Directory

I have an IIS6 application running on Windows Server 2003.

Azure App Service makes it easy to avoid the infrastructure costs associated with migrating older IIS6 applications. Microsoft has created [easy to use migration tools and detailed migration guidance](#) that enable you to check compatibility and identify any changes that need to be made. Integration with Visual Studio, TFS, and common CMS tools makes it easy to deploy IIS6 applications directly to the cloud. Once deployed, the Azure Portal provides robust management tools that enable you to scale down to manage costs and up to meet demand as necessary. With the migration tool you can:

- Quickly and easily migrate your legacy Windows Server 2003 web application to the cloud.
- Opt to leave your attached SQL database on-premises to create a hybrid application.
- Automatically move your SQL database along with your legacy application.

I'm a small business owner, and I need an inexpensive way to host my site but with future growth in mind.

Azure App Service is a great solution for this scenario, because you can start using it for free and then add more capabilities when you need them. Each free web app comes with a domain provided by Azure (*your_company.azurewebsites.net*), and the platform includes integrated deployment and management tools as well as an application gallery that make it easy to get started. There are many other services and scaling options that allow the site to evolve with increased user demand. With Azure App Service, you can:

- Begin with the free tier and then scale up as needed.
- Use the Application Gallery to quickly set up popular web applications, such as WordPress.
- Add additional Azure services and features to your application as needed.
- Secure your web app with HTTPS.

NOTE

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

I'm a web or graphic designer, and I want to design and build websites for my customers

For web developers and designers, Azure App Service integrates easily with a variety of frameworks and tools, includes deployment support for Git and FTP, and offers tight integration with tools and services such as Visual Studio and SQL Database. With App Service, you can:

- Use command-line tools for [automated tasks](#).
- Work with popular languages such as [.Net](#), [PHP](#), [Node.js](#), and [Python](#).
- Select three different scaling levels for scaling up to very high capacities.
- Integrate with other Azure services, such as [SQL Database](#), [Service Bus](#) and [Storage](#), or partner offerings from the [Azure Store](#), such as MySQL and MongoDB.
- Integrate with tools such as Visual Studio, Git, WebMatrix, WebDeploy, TFS, and FTP.

I'm migrating my multi-tier application with a web front-end to the Cloud

If you're running a multi-tier application, such as a web server that connects to a database, Azure App Service is a good option that offers tight integration with Azure SQL Database. And you can use the WebJobs feature for running backend processes.

Choose Service Fabric for one or more of your tiers if you need more control over the server environment, such as the ability to remote into your server or configure server startup tasks.

Choose Virtual Machines for one or more of your tiers if you want to use your own machine image or run server software or services that you can't configure on Service Fabric.

My application depends on highly customized Windows or Linux environments and I want to move it to the cloud.

If your application requires complex installation or configuration of software and the operating system, Virtual Machines is probably the best solution. With Virtual Machines, you can:

- Use the Virtual Machine gallery to start with an operating system, such as Windows or Linux, and then customize it for your application requirements.
- Create and upload a custom image of an existing on-premises server to run on a virtual machine in Azure.

My site uses open source software, and I want to host it in Azure

If your open source framework is supported on App Service, the languages and frameworks needed by your application are configured for you automatically. App Service enables you to:

- Use many popular open source languages, such as [.NET](#), [PHP](#), [Node.js](#), and [Python](#).
- Set up WordPress, Drupal, Umbraco, DNN, and many other third-party web applications.
- Migrate an existing application or create a new one from the Application Gallery.

If your open source framework is not supported on App Service, you can run it on one of the other Azure web hosting options. With Virtual Machines, you install and configure the software on the machine image, which can be Windows or Linux-based.

I have a line-of-business application that needs to connect to the corporate network

If you want to create a line-of-business application, your website might require direct access to services or data on the corporate network. This is possible on App Service, Service Fabric, and Virtual Machines using the [Azure Virtual Network service](#). On App Service you can use the [VNET integration feature](#), which allows your Azure applications to

run as if they were on your corporate network.

I want to host a REST API or web service for mobile clients

HTTP-based web services enable you to support a wide variety of clients, including mobile clients. Frameworks like ASP.NET Web API integrate with Visual Studio to make it easier to create and consume REST services. These services are exposed from a web endpoint, so it is possible to use any web hosting technique on Azure to support this scenario. However, App Service is a great choice for hosting REST APIs. With App Service, you can:

- Quickly create a [mobile app](#) or API app to host the HTTP web service in one of Azure's globally distributed datacenters.
- Migrate existing services or create new ones.
- Achieve SLA for availability with a single instance, or scale out to multiple dedicated machines.
- Use the published site to provide REST APIs to any HTTP clients, including mobile clients.

NOTE

If you want to get started with Azure App Service before signing up for an account, go to <https://trywebsites.azurewebsites.net>, where you can immediately create a short-lived starter app in Azure App Service for free. No credit card required, no commitments.

Next Steps

For more information about the three web hosting options, see [Introducing Azure](#).

To get started with the chosen options for your application, see the following resources:

- [Azure App Service](#)
- [Azure Cloud Services](#)
- [Azure Virtual Machines](#)
- [Service Fabric](#)

Create an ASP.NET Core web app in Azure

12/14/2017 • 4 min 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 Web Apps](#) provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy your first ASP.NET Core web app to Azure Web Apps. When you're finished, you'll have a resource group that consists of an App Service plan and an Azure web app with a deployed web application.

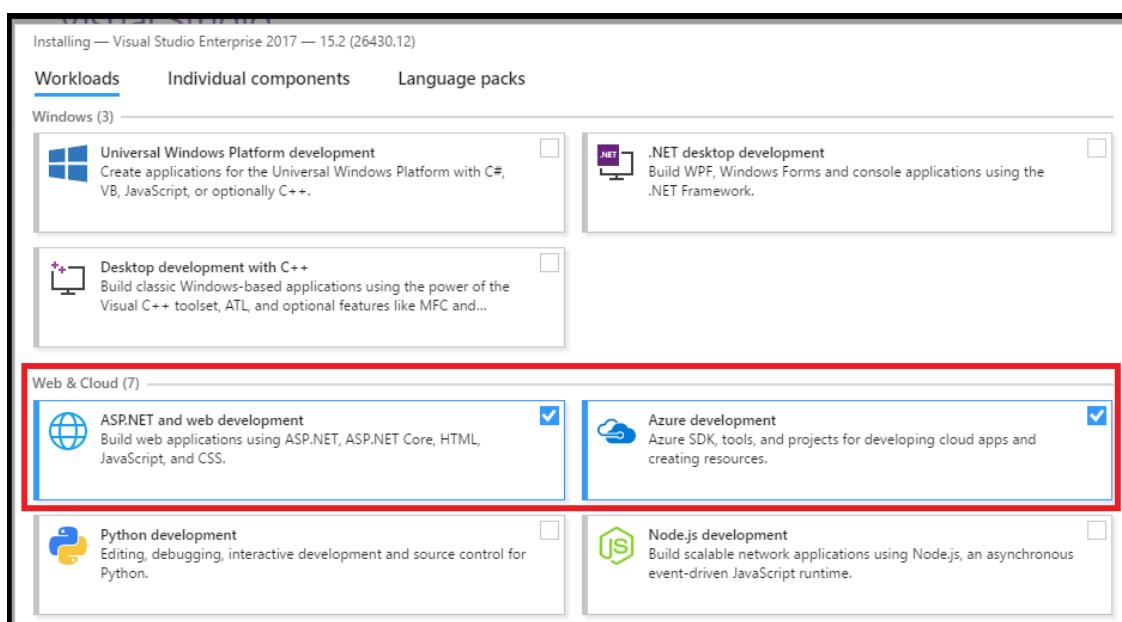
NOTE

If you're looking for how to build and deploy an ASP.NET Framework Web App, that article is available [here](#).

Prerequisites

To complete this tutorial:

- Install [Visual Studio 2017](#) with the following workloads:
 - **ASP.NET and web development**
 - **Azure development**



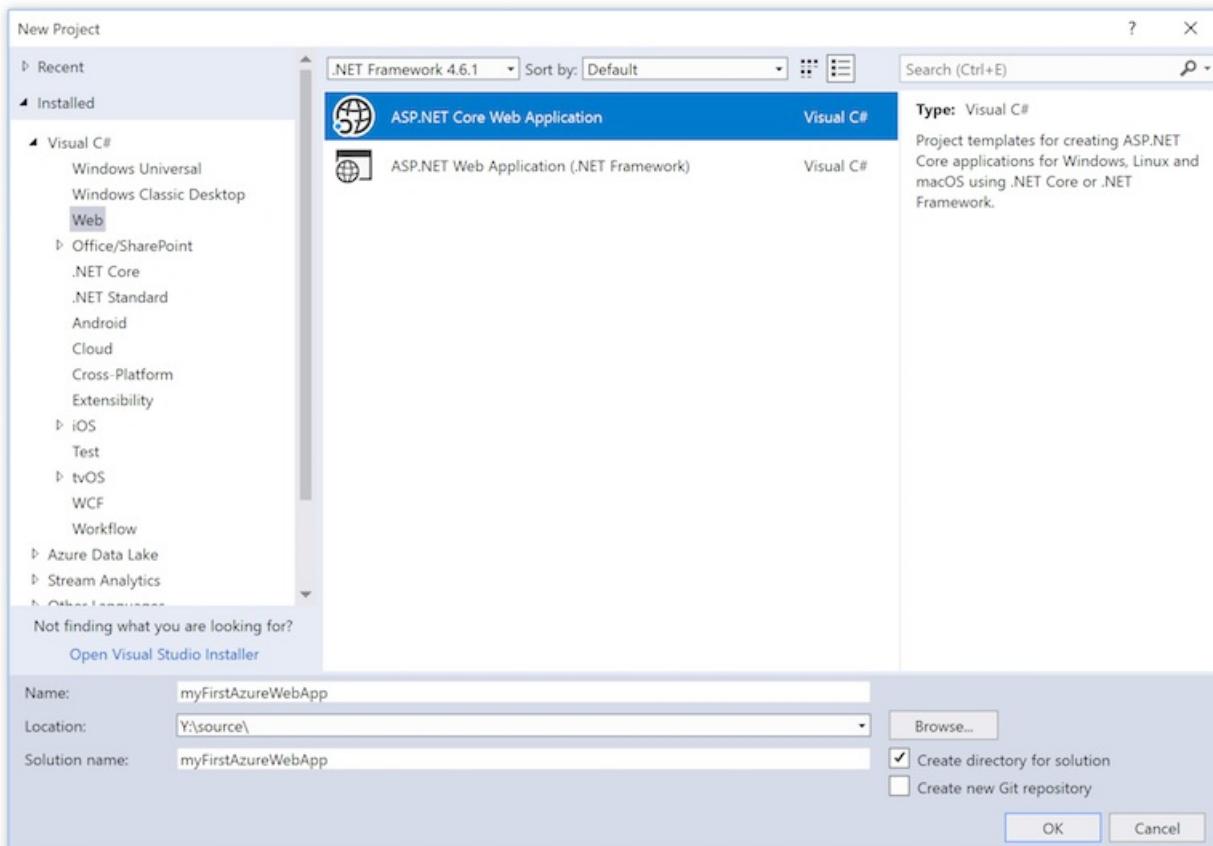
If you don't have an Azure subscription, create a [free account](#) before you begin.

Create an ASP.NET Core web app

In Visual Studio, create a project by selecting **File > New > Project**.

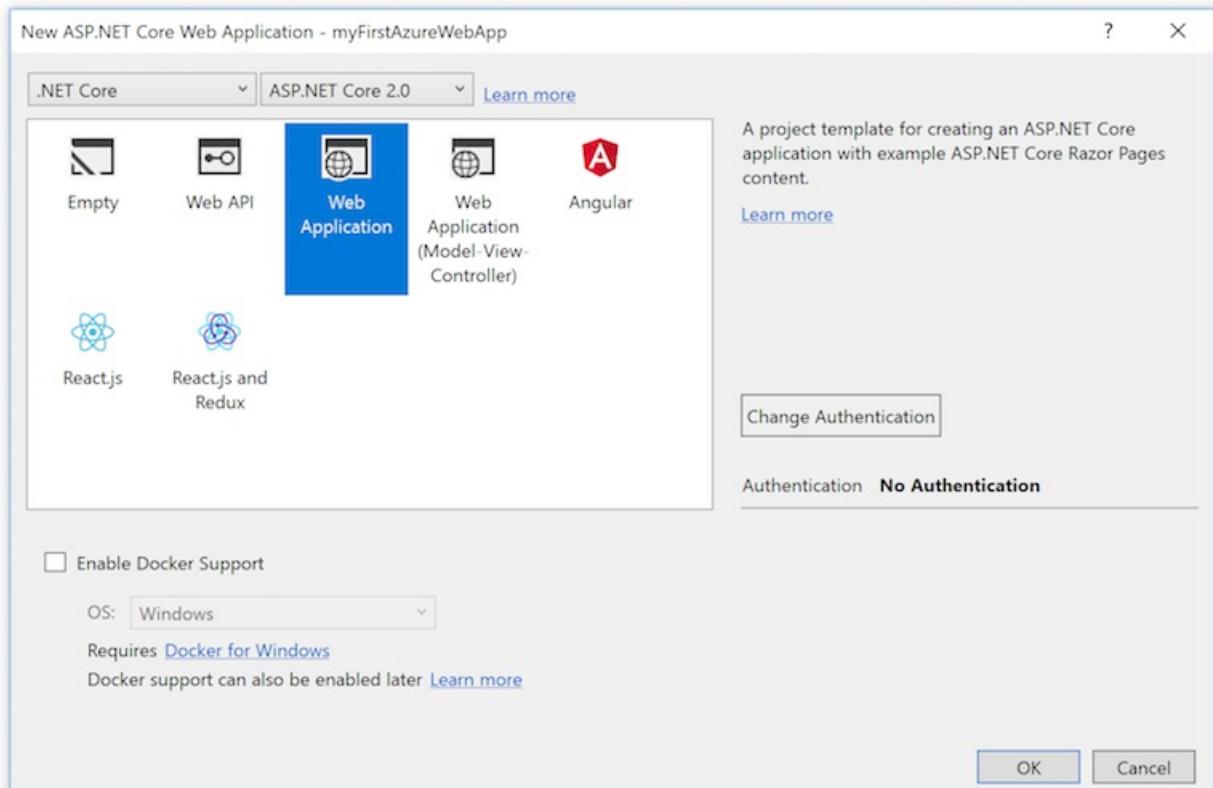
In the **New Project** dialog, select **Visual C# > Web > ASP.NET Core Web Application**.

Name the application *myFirstAzureWebApp*, and then select **OK**.

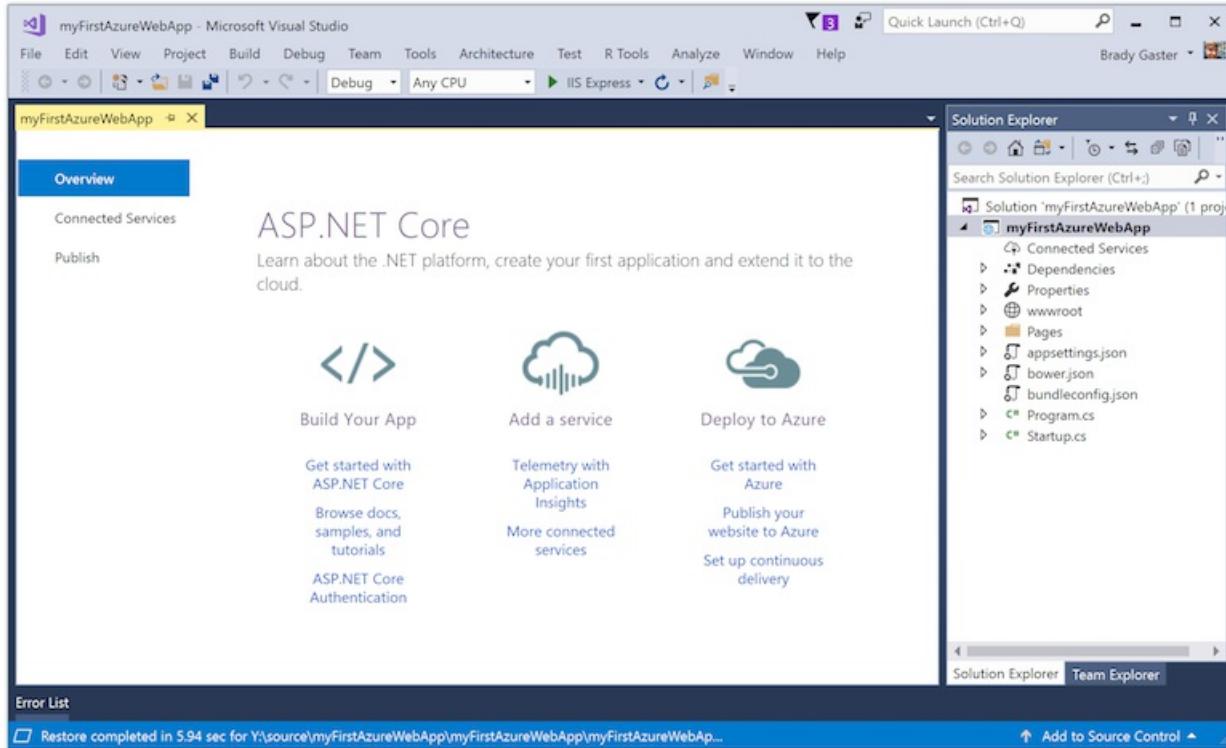


You can deploy any type of ASP.NET Core web app to Azure. For this quickstart, select the **Web Application** template, and make sure authentication is set to **No Authentication**.

Select **OK**.



Once the ASP.NET Core project is created, the ASP.NET Core welcome page will be displayed, providing numerous links to resources to help you get started.

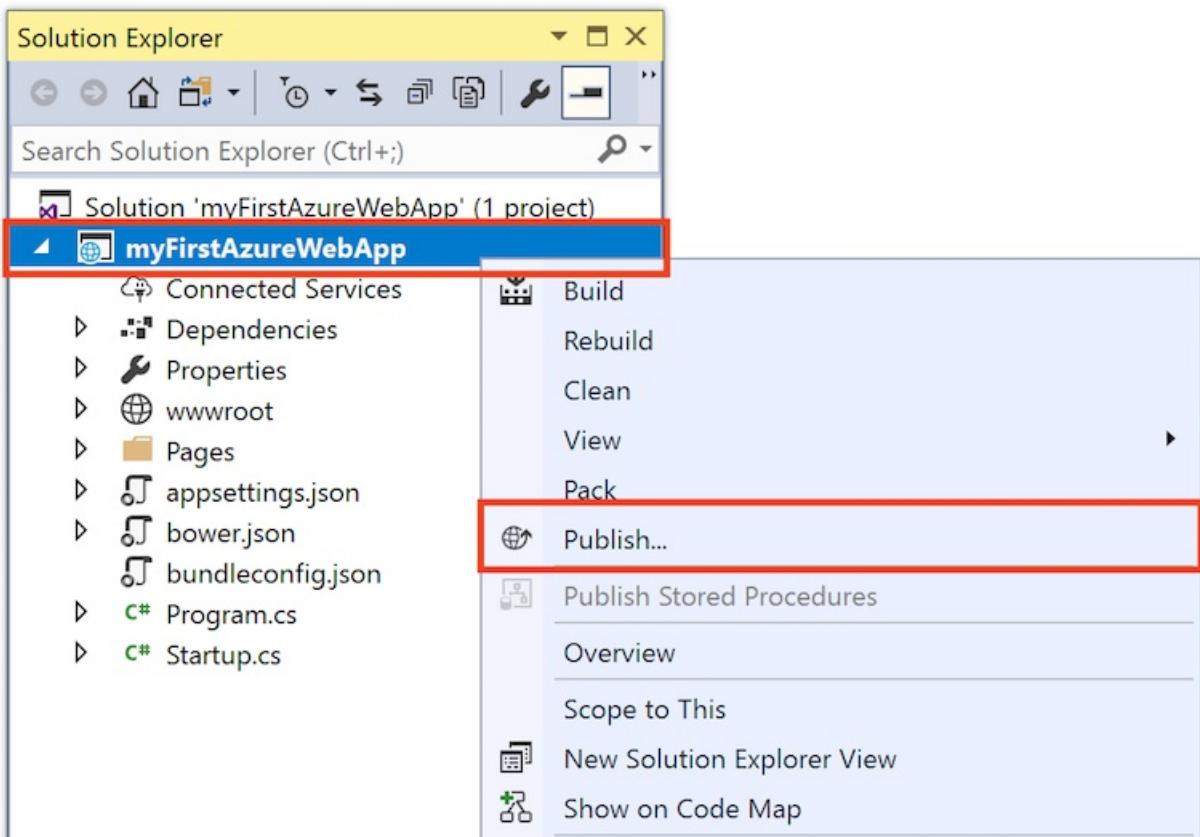


From the menu, select **Debug > Start without Debugging** to run the web app locally.

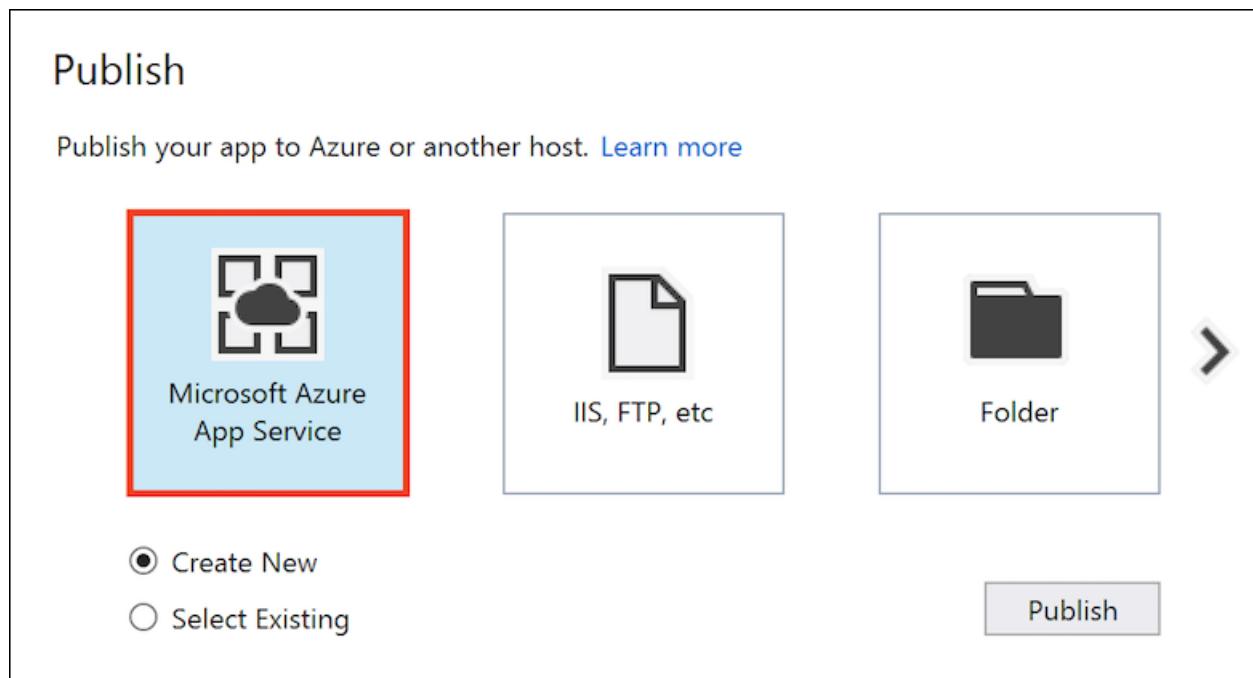
A screenshot of a web browser window titled "Home page - myFirstAz". The address bar shows "localhost:50845". The page content includes a header with the project name "myFirstAzureWebApp" and navigation links for "Home", "About", and "Contact". Below the header is a slide show with three images: "Windows", "Linux", and "OSX". The first slide has the text "Learn how to build ASP.NET apps that can run anywhere." and a "Learn More" button. The footer contains four sections: "Application uses", "How to", "Overview", and "Run & Deploy", each with a bulleted list of topics. At the bottom left, there is a copyright notice: "© 2017 - myFirstAzureWebApp".

Publish to Azure

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



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



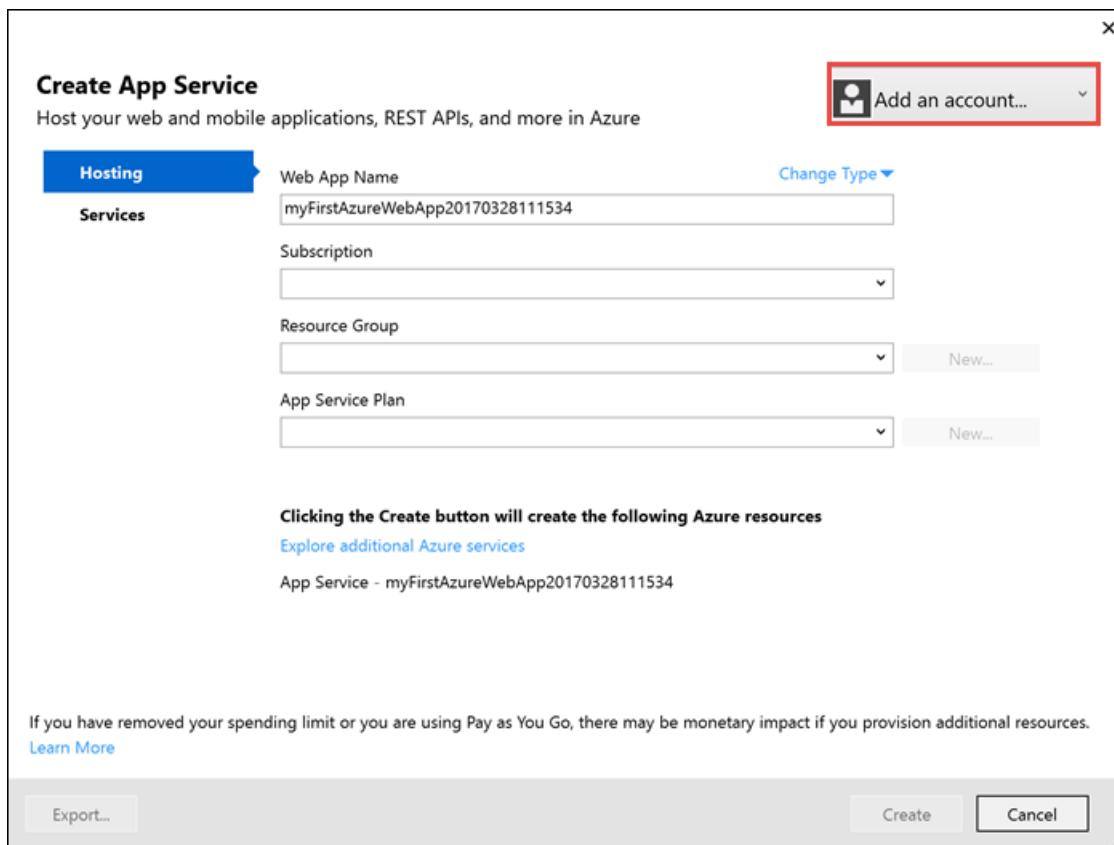
This opens the **Create App Service** dialog, which helps you create all the necessary Azure resources to run the ASP.NET Core web app in Azure.

Sign in to Azure

In the **Create App Service** 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.

NOTE

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



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.

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

Name the resource group **myResourceGroup** and select **OK**.

Create an App Service plan

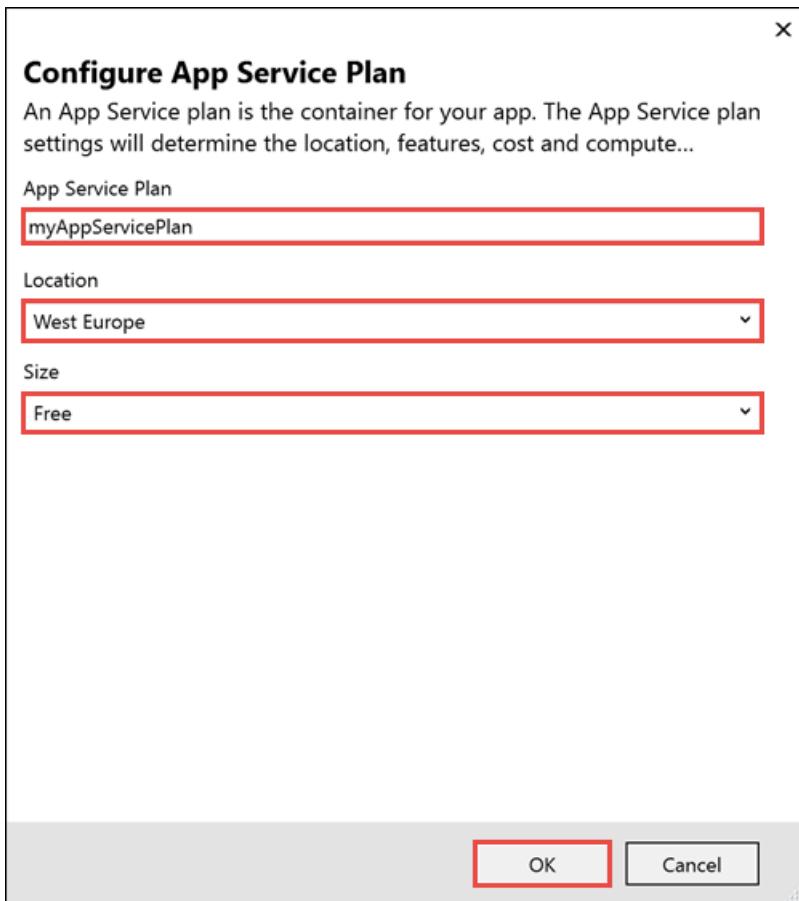
An **App Service plan** specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

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**, select **New**.

In the **Configure App Service Plan** dialog, use the settings in the table following the screenshot.



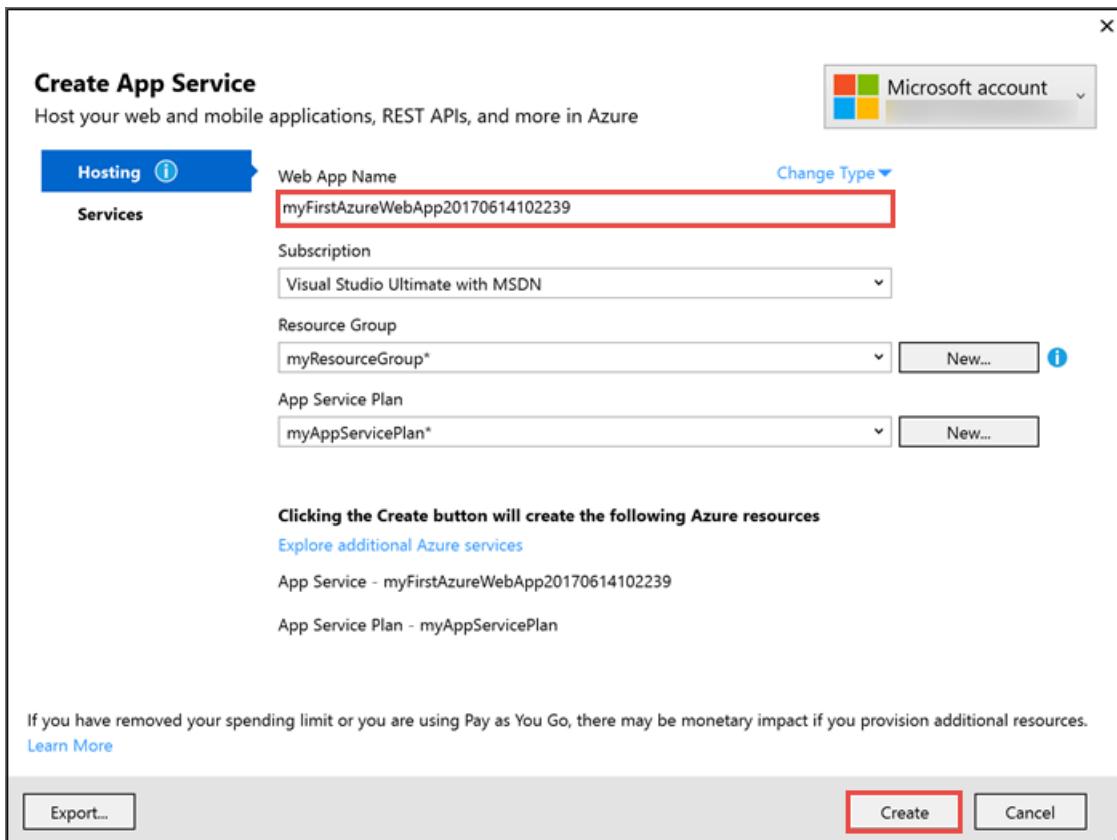
SETTING	SUGGESTED VALUE	DESCRIPTION
App Service Plan	myAppServicePlan	Name of the App Service plan.
Location	West Europe	The datacenter where the web app is hosted.
Size	Free	Pricing tier determines hosting features.

Select **OK**.

Create and publish the web app

In **Web App Name**, type a unique app name (valid characters are a-z, 0-9, and -), or accept the automatically generated unique name. The URL of the web app is `http://<app_name>.azurewebsites.net`, where <app_name> is your web app name.

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.

Application uses	How to	Overview	Run & Deploy
<ul style="list-style-type: none"> Sample pages using ASP.NET Core Razor Pages Bower for managing client-side libraries Theming using Bootstrap 	<ul style="list-style-type: none"> Working with Razor Pages. Manage User Secrets using Secret Manager. Use logging to log a message. Add packages using NuGet. Add client packages using Bower. Target development, staging or production environment. 	<ul style="list-style-type: none"> Conceptual overview of what is ASP.NET Core Fundamentals of ASP.NET Core such as Startup and middleware. Working with Data Security Client side development Develop on different platforms Read more on the documentation site 	<ul style="list-style-type: none"> Run your app Run tools such as EF migrations and more Publish to Microsoft Azure App Service

© 2017 - myFirstAzureWebApp

The web app name specified in the [create and publish step](#) 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

From the **Solution Explorer**, open *Pages/Index.cshtml*.

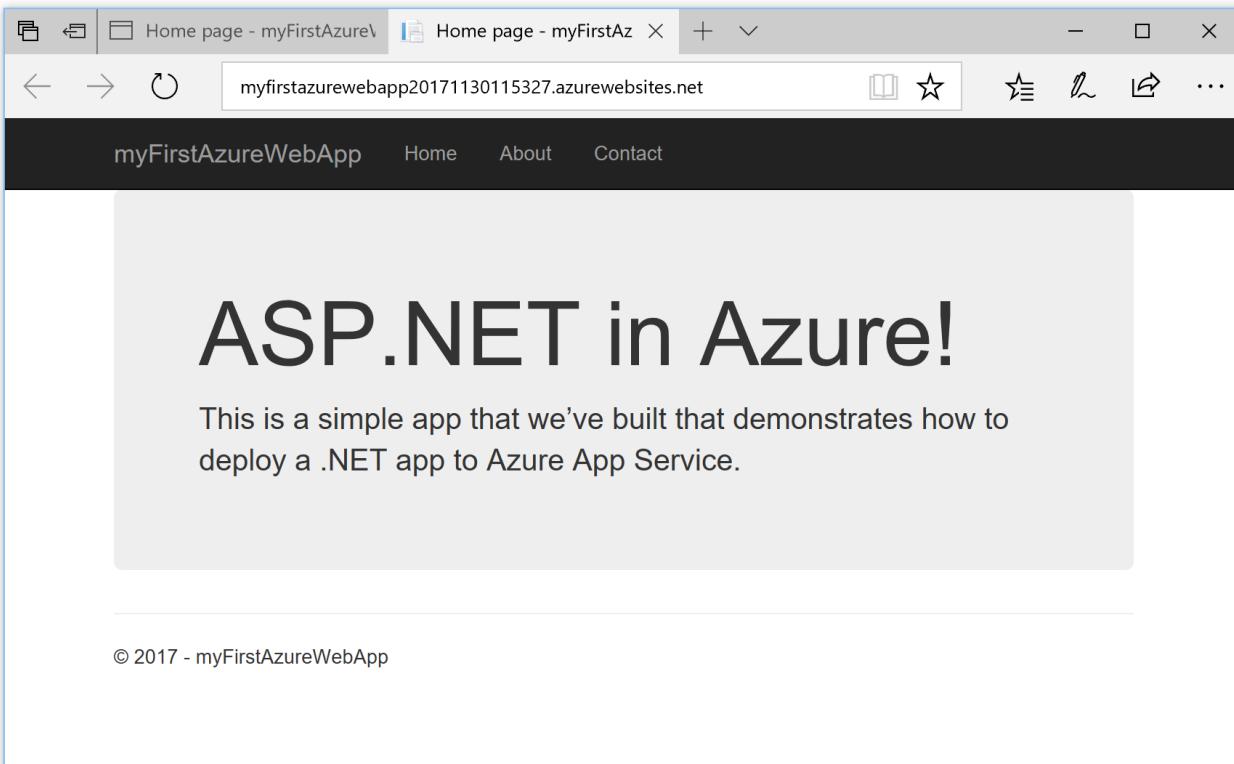
Find the `<div id="myCarousel" class="carousel slide" data-ride="carousel" data-interval="6000">` HTML tag near the top, and replace the entire element with the following code:

```
<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>
```

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

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

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



Manage the Azure web app

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

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

The screenshot shows the Microsoft Azure App Services blade. On the left, there's a sidebar with icons for different services. The main area displays a table of web apps. One specific app, 'myFirstAzureWebApp20170614102239', is highlighted with a red border. The table columns include NAME, STATUS, APP TYPE, APP SERVICE PLAN, LOCATION, and SUBSCRIPTION.

NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCRIPTION
myFirstAzureWebApp20170614102239	Running	Web app	myAppServicePlan	West Europe	Visual Studio Ultimate with ...

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

The screenshot shows the Azure App Service Overview page for 'myFirstAzureWebApp20170614102239'. The left sidebar has a navigation menu with items like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview), Application settings, and Application settings. The main content area shows the app's status as Running in West Europe, with a URL of <http://myfirstazurewebapp20170614102239>. It also lists the App Service plan as myAppServicePlan (Free) and the Subscription ID. Below this, there are two monitoring sections: 'Http 5xx' and 'Data In'. The 'Http 5xx' section shows a timeline from 10 PM to 10:30 PM with error counts for 100, 80, 60, 40, 20, and 0. The 'Data In' section shows a timeline from 10 PM to 10:15 PM with data transfer rates of 25MB, 20MB, 15MB, 10MB, 5MB, and 0MB.

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 left menu in the Azure portal, select **Resource groups** and then select **myResourceGroup**.

On the resource group 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 with SQL Database](#)

Create an ASP.NET Framework web app in Azure

12/4/2017 • 4 min to read • [Edit Online](#)

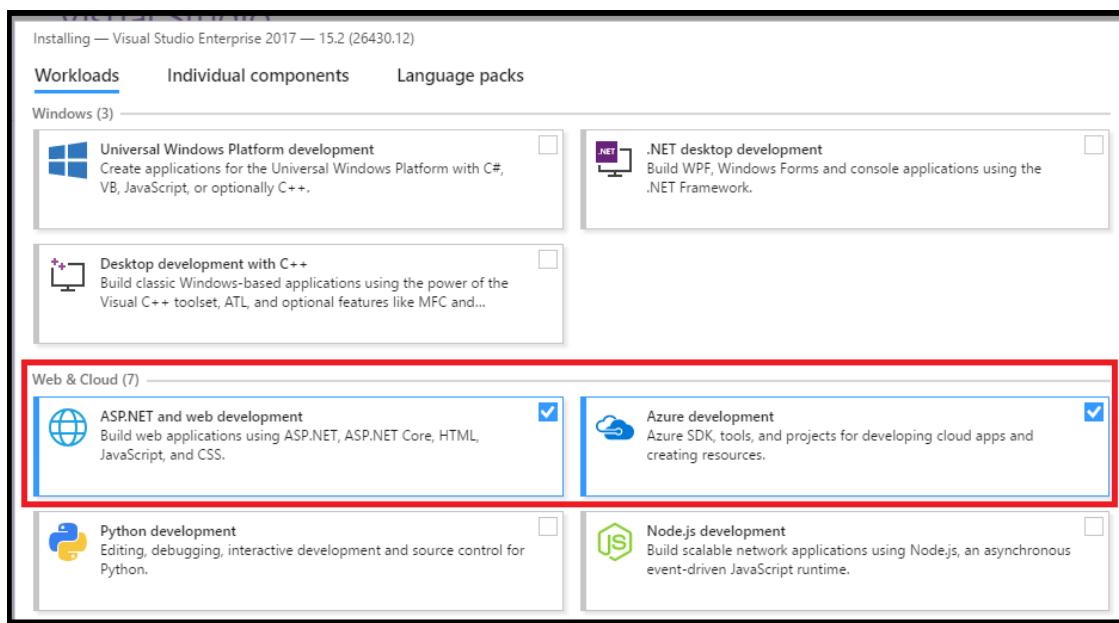
[Azure Web Apps](#) provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy your first ASP.NET web app to Azure Web Apps. When you're finished, you'll have a resource group that consists of an App Service plan and an Azure web app with a deployed web application.

Watch the video to see this quickstart in action and then follow the steps yourself to publish your first .NET app on Azure.

Prerequisites

To complete this tutorial:

- Install [Visual Studio 2017](#) with the following workloads:
 - **ASP.NET and web development**
 - **Azure development**



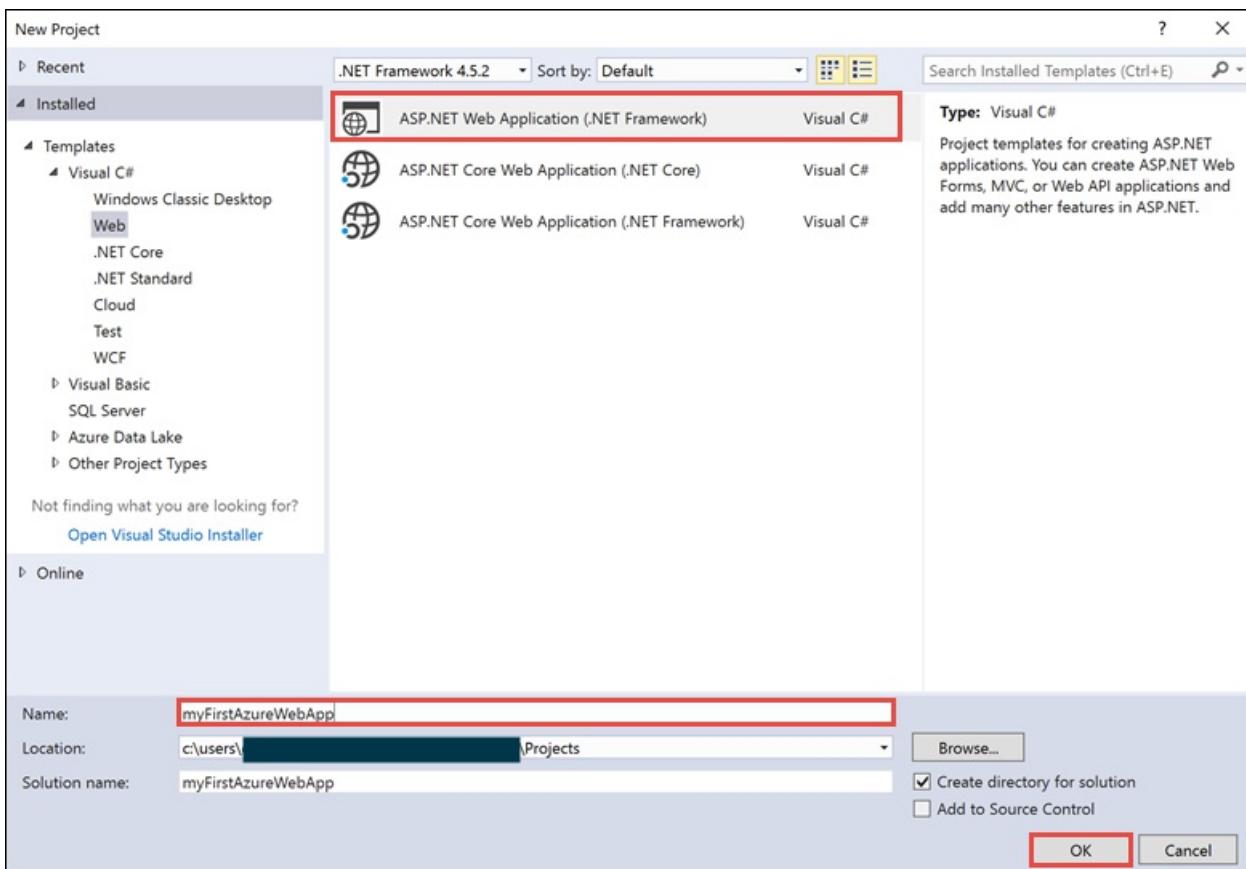
If you don't have an Azure subscription, create a [free account](#) before you begin.

Create an ASP.NET web app

In Visual Studio, create a project by selecting **File > New > Project**.

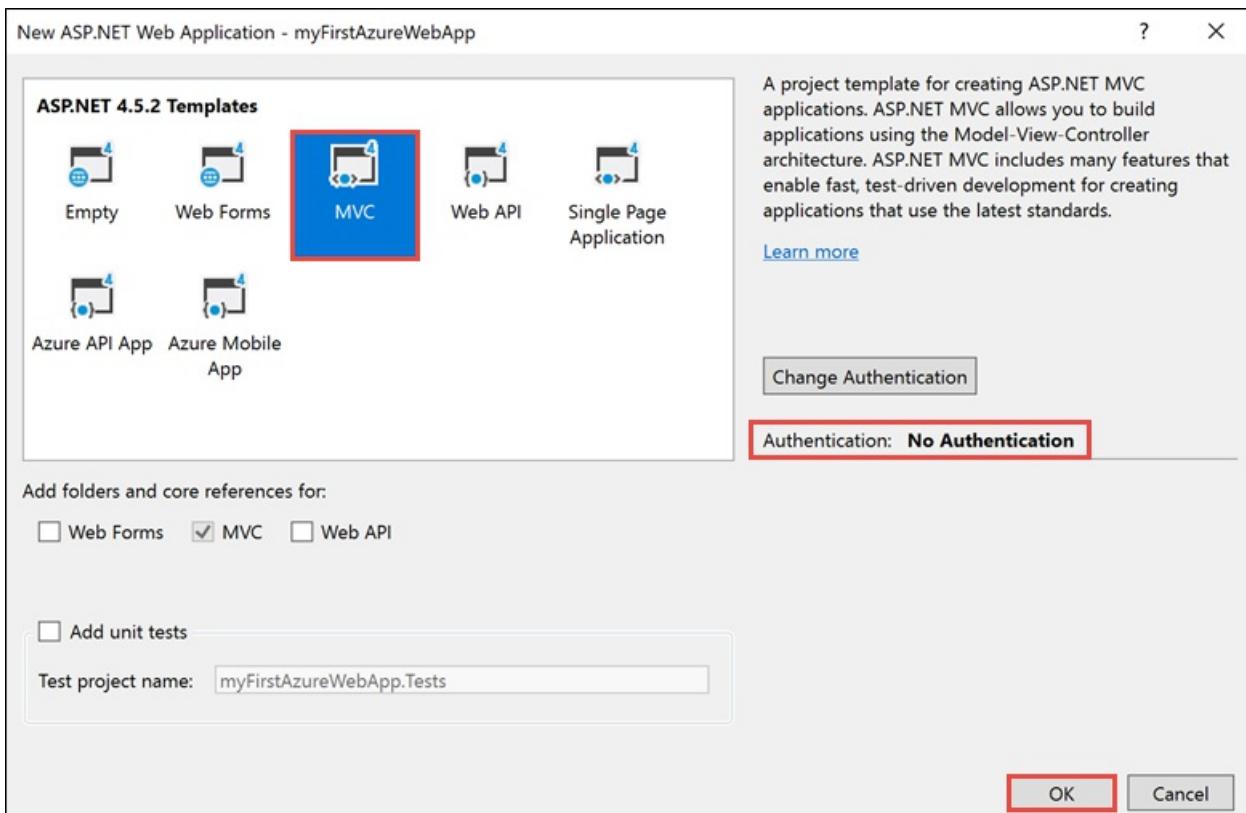
In the **New Project** dialog, select **Visual C# > Web > ASP.NET Web Application (.NET Framework)**.

Name the application *myFirstAzureWebApp*, and then select **OK**.

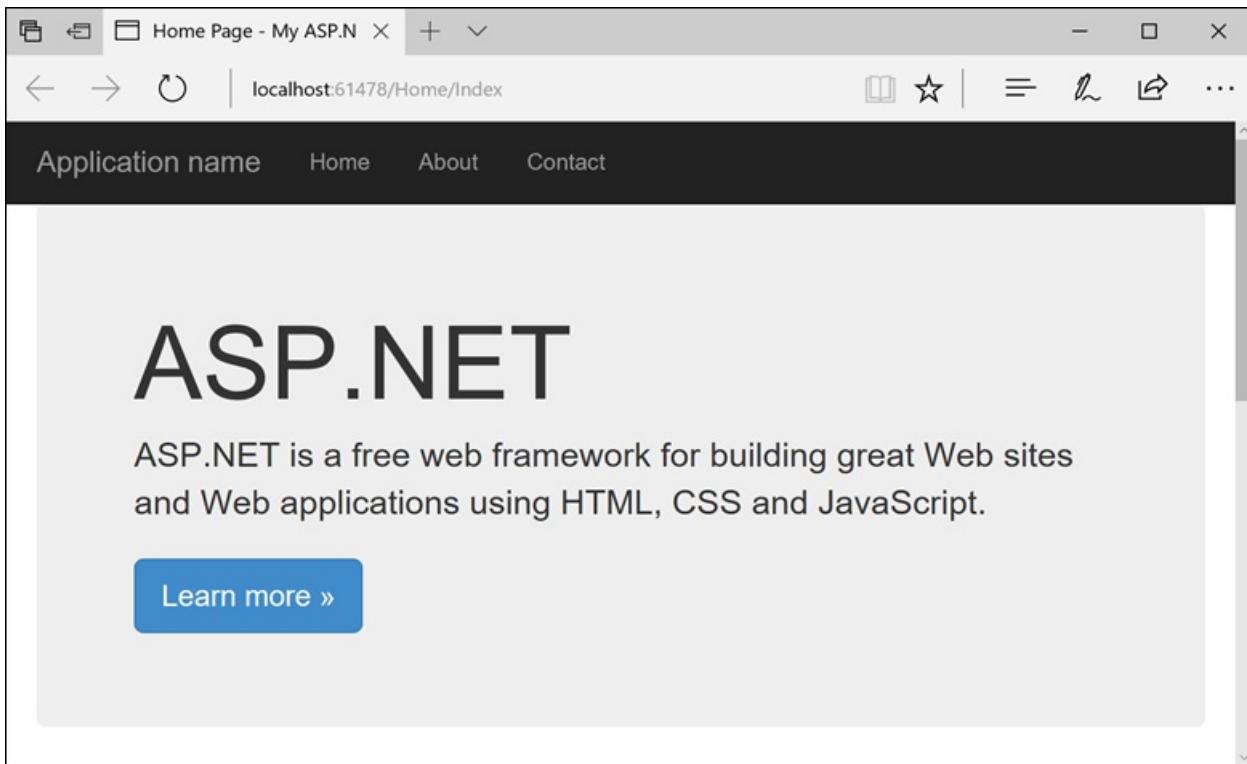


You can deploy any type of ASP.NET web app to Azure. For this quickstart, select the **MVC** template, and make sure authentication is set to **No Authentication**.

Select **OK**.

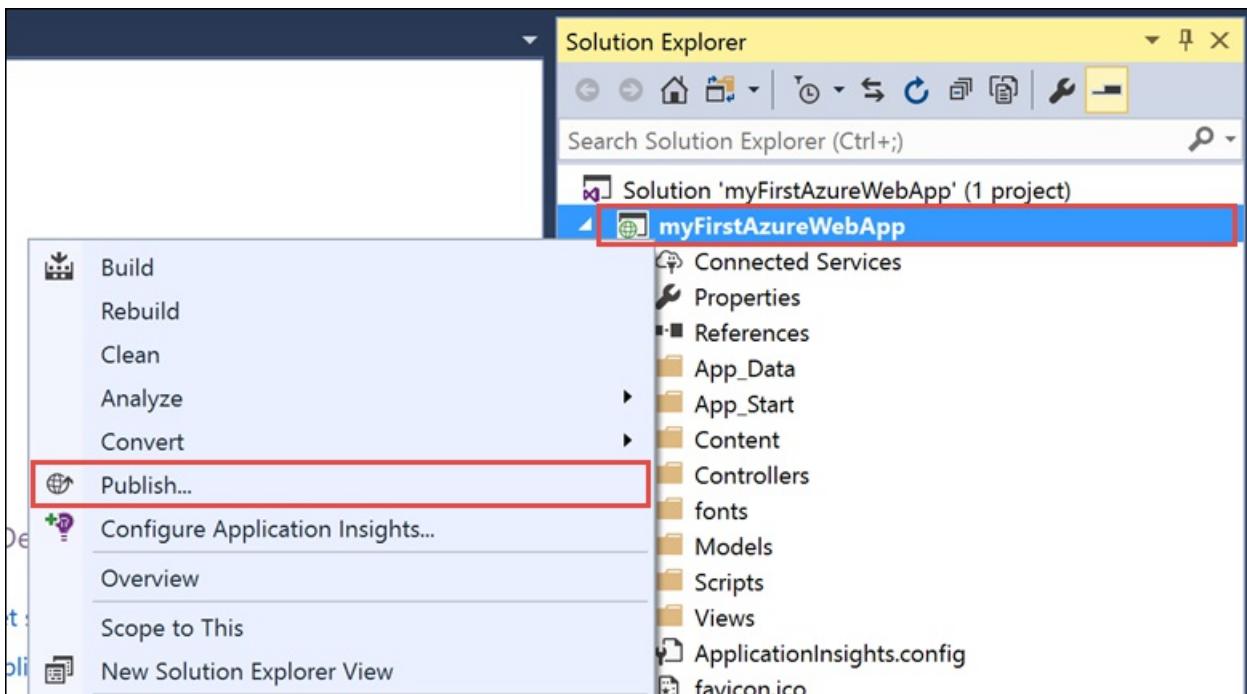


From the menu, select **Debug > Start without Debugging** to run the web app locally.



Publish to Azure

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



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

Publish

Publish your app to Azure or another host. [Learn more](#)



This opens the **Create App Service** dialog, which helps you create all the necessary Azure resources to run the ASP.NET web app in Azure.

Sign in to Azure

In the **Create App Service** 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.

NOTE

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

The image shows the 'Create App Service' dialog box. On the left, there's a 'Hosting Services' section with tabs for 'Hosting' (selected) and 'Services'. The 'Web App Name' field contains 'myFirstAzureWebApp20170328111534'. To the right of the name is a 'Change Type' dropdown. Below the name are fields for 'Subscription' (a dropdown menu), 'Resource Group' (a dropdown menu with a 'New...' button), and 'App Service Plan' (a dropdown menu with a 'New...' button). At the top right is a 'Add an account...' button with a user icon, which is highlighted with a red border. At the bottom, there's a note about creating resources, a link to explore services, and a note about spending limits. The 'Create' and 'Cancel' buttons are at the very bottom.

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.

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

Name the resource group **myResourceGroup** and select **OK**.

Create an App Service plan

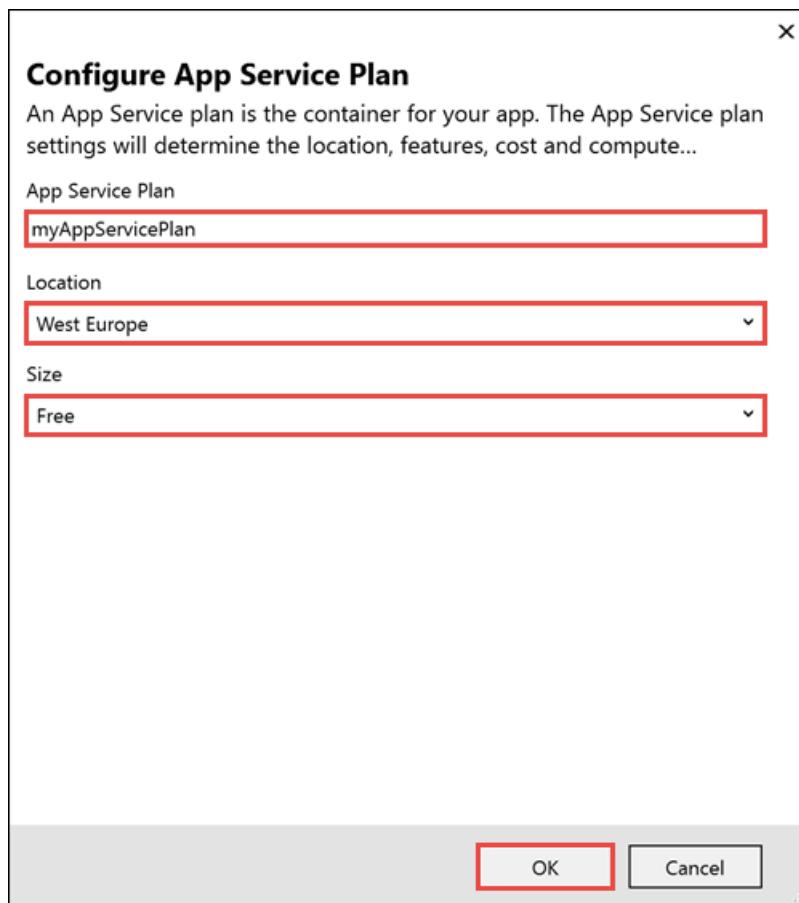
An **App Service plan** specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

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**, select **New**.

In the **Configure App Service Plan** dialog, use the settings in the table following the screenshot.



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.

Select **OK**.

Create and publish the web app

In **Web App Name**, type a unique app name (valid characters are `a-z`, `0-9`, and `-`), or accept the automatically generated unique name. The URL of the web app is `http://<app_name>.azurewebsites.net`, where `<app_name>` is your web app name.

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

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

Hosting i **Services**

Web App Name: myFirstAzureWebApp20170614102239 Change Type ▾

Subscription: Visual Studio Ultimate with MSDN

Resource Group: myResourceGroup* New... i

App Service Plan: myAppServicePlan* New...

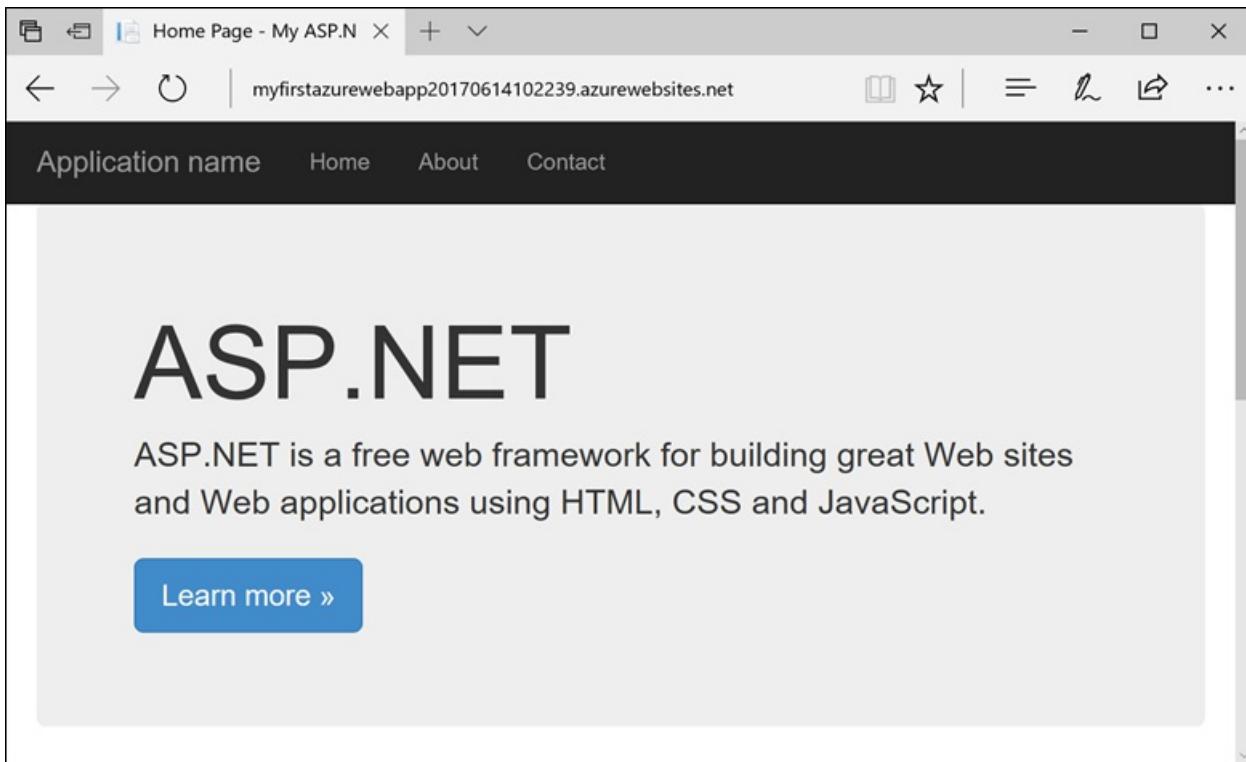
Clicking the Create button will create the following Azure resources
[Explore additional Azure services](#)

App Service - myFirstAzureWebApp20170614102239
App Service Plan - myAppServicePlan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
[Learn More](#)

Export... Create Cancel

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



The web app name specified in the [create and publish step](#) 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

From the **Solution Explorer**, open `Views\Home\Index.cshtml`.

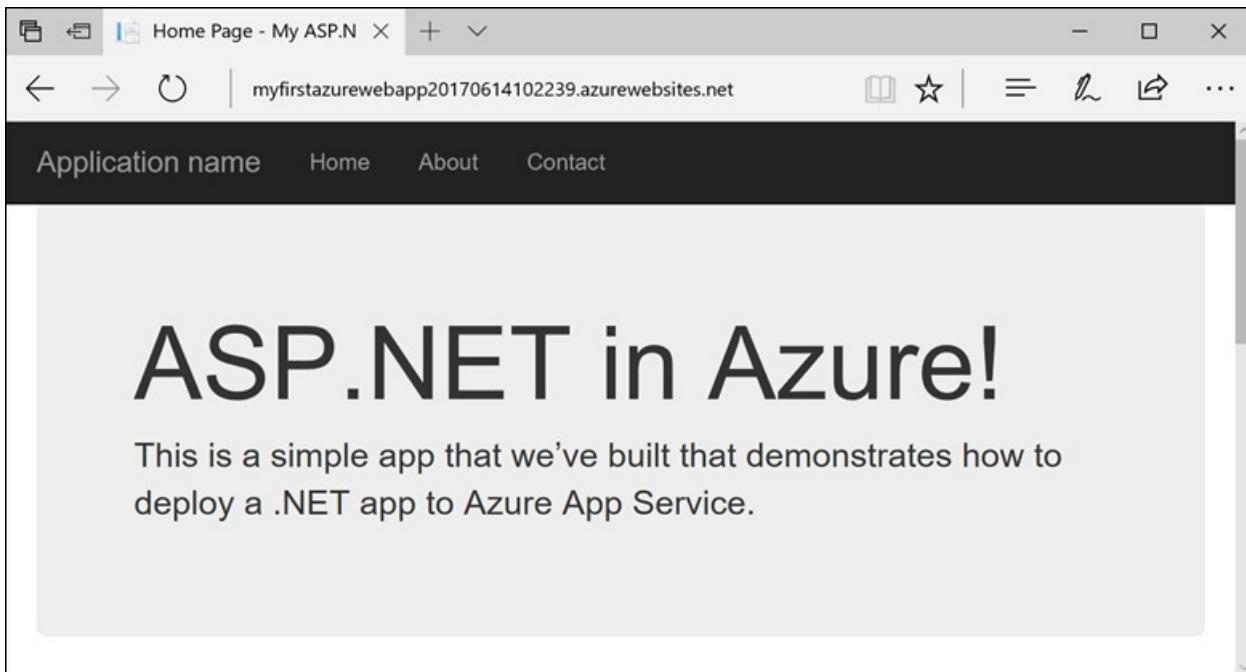
Find the `<div class="jumbotron">` HTML tag near the top, and replace the entire element with the following code:

```
<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>
```

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

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

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



Manage the Azure web app

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

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

A screenshot of the Microsoft Azure portal's "App Services" overview page. The top navigation bar includes "Microsoft Azure" and "App Services". The main area shows a table with one item: "myFirstAzureWebApp20170614102239". The "NAME" column shows the app name with a globe icon, the "STATUS" column shows "Running", the "APP TYPE" column shows "Web app", the "APP SERVICE PLAN" column shows "myAppServicePlan", the "LOCATION" column shows "West Europe", and the "SUBSCRIPTION" column shows "Visual Studio Ultimate with ...". A red box highlights the "myFirstAzureWebApp20170614102239" row. On the far left, there is a vertical sidebar with icons for different services, and the "App Services" icon is highlighted with a red box.

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

The left menu provides different pages for configuring your app.

Clean up resources

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

From the left menu in the Azure portal, select **Resource groups** and then select **myResourceGroup**.

On the resource group 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 with SQL Database](#)

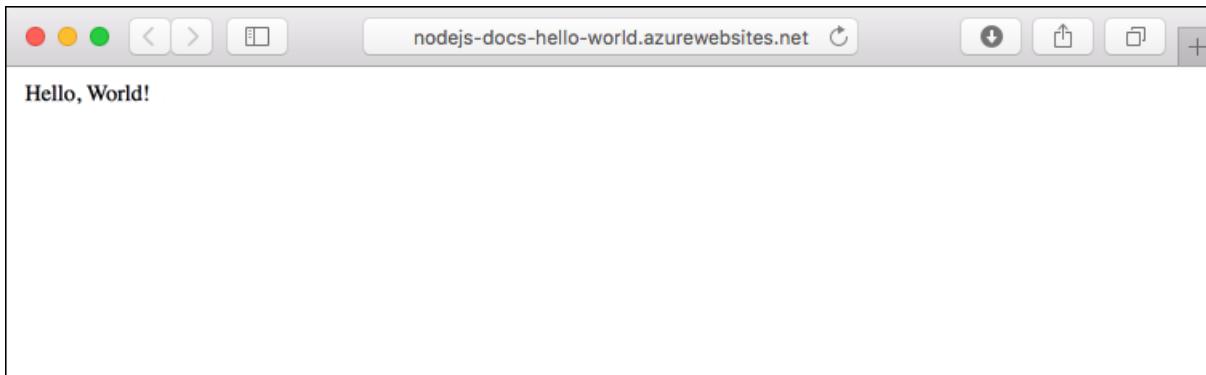
Create a Node.js web app in Azure

12/14/2017 • 6 min 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 Node.js web app in Azure App Service on Linux](#).

[Azure Web Apps](#) provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy a Node.js app to Azure Web Apps. You create the web app using the [Azure CLI](#), and you use Git to deploy sample Node.js 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.

Prerequisites

To complete this quickstart:

- [Install Node.js and NPM](#)

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

Download the sample

Download the sample Node.js project from <https://github.com/Azure-Samples/nodejs-docs-hello-world/archive/master.zip> and extract the ZIP archive.

In a terminal window, navigate to the root directory of the sample Node.js project (the one that contains *index.js*).

Run the app locally

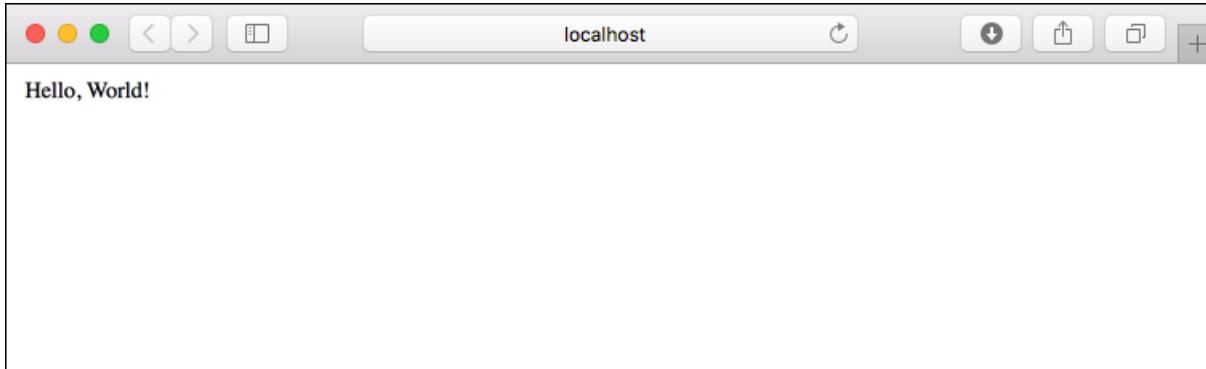
Run the application locally by opening a terminal window and using the `npm start` script to launch the built-in

Node.js HTTP server.

```
npm start
```

Open a web browser, and navigate to the sample app at <http://localhost:1337>.

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.

Create a project ZIP file

Create a ZIP archive of everything in your project. The following command uses the default tool in your terminal:

```
# Bash  
zip -r myAppFiles.zip .  
  
# PowerShell  
Compress-Archive -Path * -DestinationPath myAppFiles.zip
```

Later, you upload this ZIP file to Azure and deploy it to App Service.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	A screenshot of a code block. In the top right corner, there is a button labeled "Try It" with a red border around it. To its left is the text "Azure CLI". Below the button are several small icons.
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	A screenshot of the Azure portal's navigation bar. The "Cloud Shell" icon, which looks like a terminal window, is highlighted with a red border. Other icons include a magnifying glass, a bell, and a gear.

Upload the ZIP file

In the [Azure portal](#), click **Resource groups** > **cloud-shell-storage-<your_region>** > **<storage_account_name>**.

The screenshot shows the Azure portal interface. On the left, the 'Resource groups' blade is open, displaying a list of resource groups: 'cloud-shell-storage-westeurope...', 'domain', 'myResourceGroup', and 'Reimbu'. The 'cloud-shell-storage-westeurope...' group is selected and highlighted with a red box. On the right, the 'cloud-shell-storage-westeurope' resource group details page is shown. It includes sections for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'SETTINGS', 'Quickstart', 'Resource costs', 'Deployments', 'Policies', 'Properties', 'Locks', and 'Automation script'. At the top, it shows the subscription name 'Visual Studio Ultimate with MSON', deployment status 'No deployments', and a table listing one storage account resource.

In the **Overview** page of the storage account, select **Files**.

Select the automatically generated file share and select **Upload**. This file share is mounted in the Cloud Shell as `clouddrive`.

The screenshot shows the 'File service' blade. On the left, the 'File share' section lists a single share named 'cloudconsole'. On the right, the 'File share' details page shows a table with one entry: 'cloudconsole' (Type: Directory). At the top, there are buttons for 'Connect', 'Upload' (which is highlighted with a red box), 'Add directory', 'Refresh', 'Delete share', 'Properties', 'Quota', and 'Snapshot'. A search bar at the top right is also visible.

Click the file selector and select your ZIP file, then click **Upload**.

In the Cloud Shell, use `ls` to verify that you can see the uploaded ZIP file in the default `clouddrive` share.

```
ls clouddrive
```

Create a resource group

In the Cloud Shell, create a resource group with the `az group create` command.

A **resource group** is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group named `myResourceGroup` in the *West Europe* location. To see all supported locations for App Service, run the `az appservice list-locations` command.

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

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

Create an Azure App Service plan

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

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)

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

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 `NODE|6.9`. To see all supported runtimes, run [az webapp list-runtimes](#).

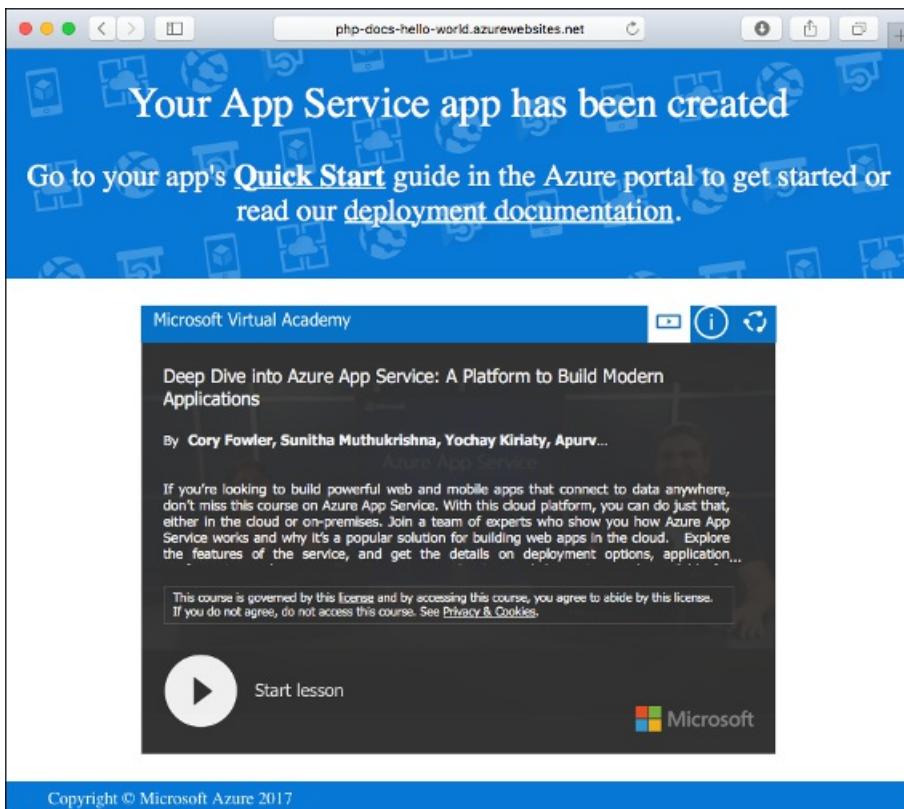
```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime
"NODE|6.9"
```

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

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

Browse to your newly created web app. Replace <app name> with a unique app name.

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



Deploy uploaded ZIP file

In the Cloud Shell, deploy the uploaded ZIP file to your web app by using the [az webapp deployment source config-zip](#) command. Be sure to replace <app_name> with the name of your web app.

```
az webapp deployment source config-zip --resource-group myResourceGroup --name <app_name> --src clouddrive/myAppFiles.zip
```

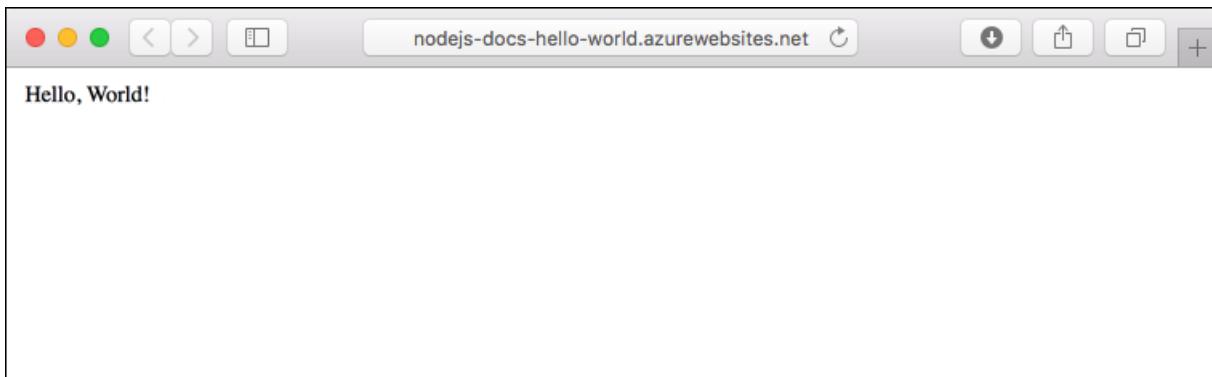
This command deploys the files and directories from the ZIP file to your default App Service application folder (`\home\site\wwwroot`) and restarts the app. If any additional custom build process is configured, it is run as well.

Browse to the app

Browse to the deployed application using your web browser.

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

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



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

Update and redeploy the code

Using a text editor, open the `index.js` file in the Node.js app, and make a small change to the text in the call to `response.end`:

```
response.end("Hello Azure!");
```

In the local terminal window, navigate to your application's root directory, create a new ZIP file for your updated project.

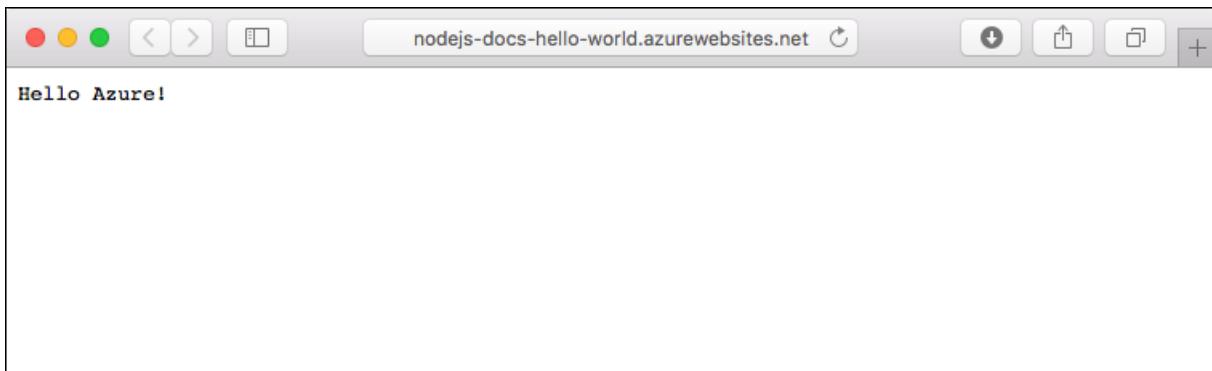
```
# Bash  
zip -r myUpdatedAppFiles.zip .  
  
# PowerShell  
Compress-Archive -Path * -DestinationPath myUpdatedAppFiles.zip
```

Upload this new ZIP file to the Cloud Shell, using the same steps in [Upload the ZIP file](#).

Then, in the Cloud Shell, deploy your uploaded ZIP file again.

```
az webapp deployment source config-zip --resource-group myResourceGroup --name <app_name> --src  
clouddrive/myUpdatedAppFiles.zip
```

Switch back to the browser window that opened in the **Browse to the app** step, and refresh the page.



Manage your new Azure web app

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

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

Microsoft Azure App Services

Subscriptions: 1 of 19 selected – Don't see a subscription? [Switch directories](#)

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

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

Microsoft Azure App Services > nodejs-docs-hello-world

nodejs-docs-hello-world App Service

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Browse Stop Swap Restart Delete More

Essentials

Resource group (change) myResourceGroup

Status Running

Location West Europe

Subscription name (change) Field Engineer Demo

Subscription ID [REDACTED]

URL <http://nodejs-docs-hello-world.azurewebsites.net>

App Service plan/pricing tier quickStartPlan (Standard: 1 Small)

FTP/deployment username nodejs-docs-hello-world\demoaccount

FTP hostname <ftp://waws-prod-am2-085.ftp.azurewebsite.net>

FTPS hostname <https://waws-prod-am2-085.ftp.azurewebsite.net>

Monitoring

Requests and errors

The left menu provides different pages for configuring your app.

Clean up resources

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

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

This command may take a minute to run.

Next steps

[Nodejs with MongoDB](#)

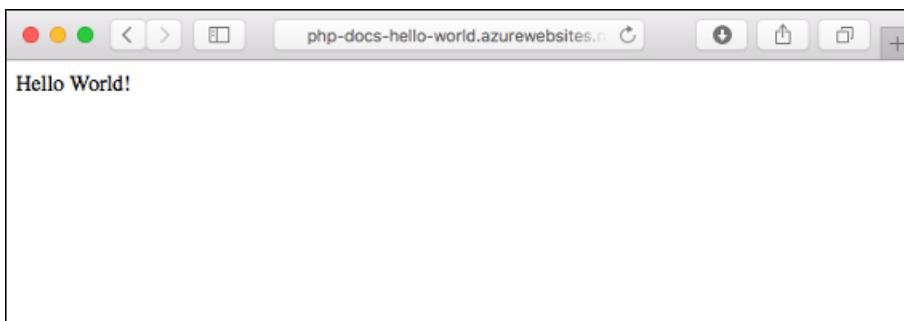
Create a PHP web app in Azure

12/14/2017 • 6 min 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 Web Apps](#) provides a highly scalable, self-patching web hosting service. This quickstart tutorial shows how to deploy a PHP app to Azure Web Apps. You create the web app using the [Azure CLI](#) in Cloud Shell, and you use a [ZIP file](#) to deploy the 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.

Prerequisites

To complete this quickstart:

- [Install PHP](#)

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

Download the sample locally

Download the sample PHP project from <https://github.com/Azure-Samples/php-docs-hello-world/archive/master.zip> and extract the ZIP archive.

In a terminal window, navigate to the root directory of the sample PHP project (the one that contains *index.php*).

Run the app locally

Run the application locally by opening a terminal window and using 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.

Create a project ZIP file

Create a ZIP archive of everything in your project. The following command uses the default tool in your terminal:

```
# Bash  
zip -r myAppFiles.zip .  
  
# PowerShell  
Compress-Archive -Path * -DestinationPath myAppFiles.zip
```

Later, you upload this ZIP file to Azure and deploy it to App Service.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	A screenshot of the Azure portal interface. On the top navigation bar, there is a series of icons. The fourth icon from the left is a blue square with a white right-pointing arrow, which is highlighted with a red rectangular box.
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	A screenshot of the Azure portal interface. On the top navigation bar, there is a series of icons. The fourth icon from the left is a blue square with a white right-pointing arrow, which is highlighted with a red rectangular box.

Upload the ZIP file

In the [Azure portal](#), click **Resource groups** > **cloud-shell-storage-<your_region>** > **<storage_account_name>**.

The screenshot shows the Azure portal interface. On the left, the 'Resource groups' blade is open, displaying a list of resource groups: 'cloud-shell-storage-westeurope...', 'domain', 'myResourceGroup', and 'Reimbu'. The 'cloud-shell-storage-westeurope...' group is selected and highlighted with a red box. On the right, the 'cloud-shell-storage-westeurope' resource group details page is shown. It includes sections for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'SETTINGS', 'Quickstart', 'Resource costs', 'Deployments', 'Policies', 'Properties', 'Locks', and 'Automation script'. At the top, it shows the subscription name 'Visual Studio Ultimate with MSON', deployment status 'No deployments', and a table listing one storage account resource.

In the **Overview** page of the storage account, select **Files**.

Select the automatically generated file share and select **Upload**. This file share is mounted in the Cloud Shell as `clouddrive`.

The screenshot shows the 'File service' blade. On the left, the 'File share' section lists a single share named 'cloudconsole'. On the right, the 'File share' details page shows a table with one entry: 'cloudconsole' (Type: Directory). At the top, there are buttons for 'Connect', 'Upload' (which is highlighted with a red box), 'Add directory', 'Refresh', 'Delete share', 'Properties', 'Quota', and 'Snapshot'. A search bar at the top right is also visible.

Click the file selector and select your ZIP file, then click **Upload**.

In the Cloud Shell, use `ls` to verify that you can see the uploaded ZIP file in the default `clouddrive` share.

```
ls clouddrive
```

Create a resource group

In the Cloud Shell, create a resource group with the `az group create` command.

A **resource group** is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group named `myResourceGroup` in the *West Europe* location. To see all supported locations for App Service, run the `az appservice list-locations` command.

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

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

Create an Azure App Service plan

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

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)

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

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](#).

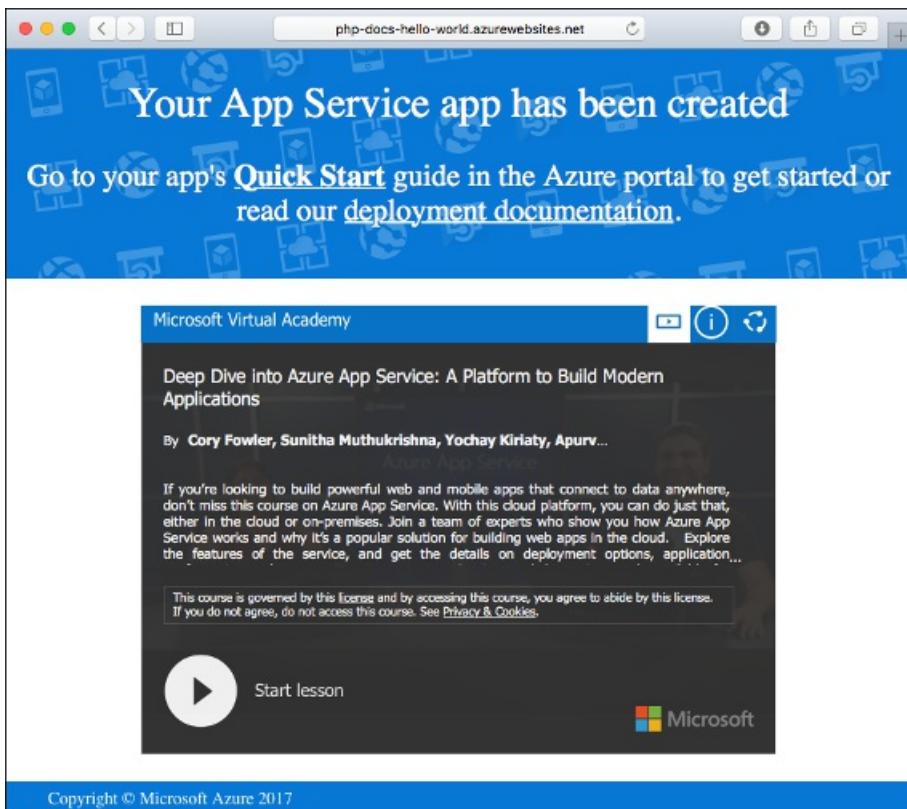
```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime "PHP|7.0"
```

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

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

Browse to your newly created web app. Replace <app name> with a unique app name.

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



Deploy uploaded ZIP file

In the Cloud Shell, deploy the uploaded ZIP file to your web app by using the [az webapp deployment source config-zip](#) command. Be sure to replace <app_name> with the name of your web app.

```
az webapp deployment source config-zip --resource-group myResourceGroup --name <app_name> --src clouddrive/myAppFiles.zip
```

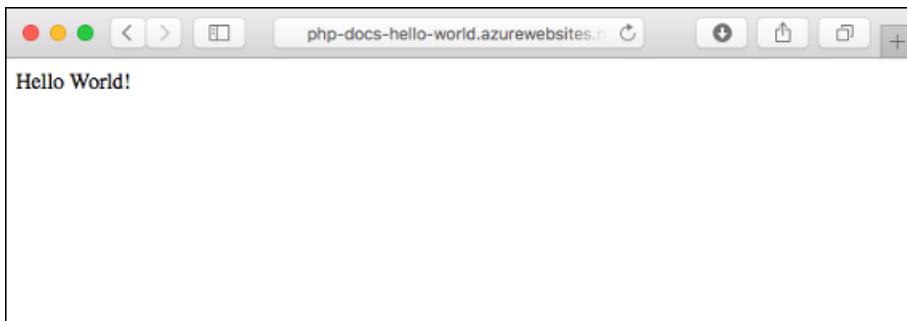
This command deploys the files and directories from the ZIP file to your default App Service application folder (`\home\site\wwwroot`) and restarts the app. If any additional custom build process is configured, it is run as well.

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, navigate to your application's root directory, create a new ZIP file for your updated project.

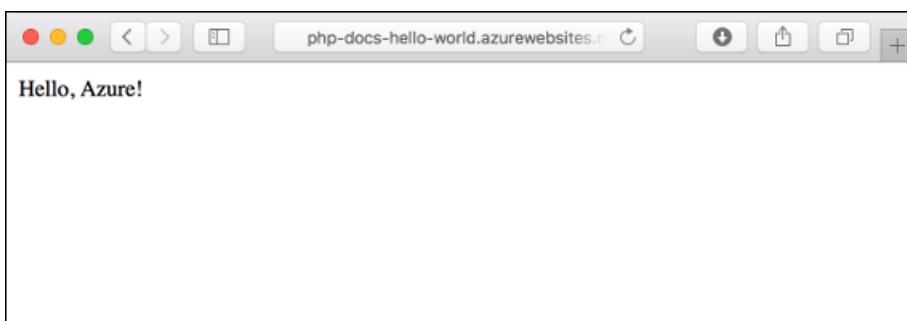
```
# Bash  
zip -r myUpdatedAppFiles.zip .  
  
# PowerShell  
Compress-Archive -Path * -DestinationPath myUpdatedAppFiles.zip
```

Upload this new ZIP file to the Cloud Shell, using the same steps in [Upload the ZIP file](#).

Then, in the Cloud Shell, deploy your uploaded ZIP file again.

```
az webapp deployment source config-zip --resource-group myResourceGroup --name <app_name> --src  
clouddrive/myUpdatedAppFiles.zip
```

Switch back to the browser window that opened in the **Browse to the app** step, and refresh the page.



Manage your new Azure web app

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

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

The screenshot shows the Microsoft Azure portal interface for App Services. The top navigation bar includes 'Microsoft Azure' and 'App Services'. The main content area displays a table with one item: 'php-docs-hello-world' under 'NAME', 'myResourceGroup' under 'RESOURCE GROUP', 'West Europe' under 'LOCATION', and 'Standard: 1 Small' under 'APP SERVICE PLAN'. A sidebar on the left lists various icons for managing resources.

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

This screenshot shows the detailed configuration page for the 'php-docs-hello-world' app. The left sidebar has a 'Search (Ctrl+I)' field and links for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Deployment' (with 'Quickstart'), and 'Monitoring'. The main panel is titled 'Essentials' and contains the following details:

Resource group [change]	URL
myResourceGroup	http://php-docs-hello-world.azurewebsites.net
Status	Running
Location	West Europe
Subscription name [mgr]	quickStartPlan (Standard: 1 Small)
Subscription ID	Sd6c94cd-6781-43e3-8a94-ceef4c28850e
Git/Deployment username	demopaccount
Git clone url	https://demopaccount@php-docs-hello-world.scm.azurewebsites.net:443/
FTP hostname	ftp://waws-prod-am2-085.ftp.azurewebsites.net

The left menu provides different pages for configuring your app.

Clean up resources

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

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

This command may take a minute to run.

Next steps

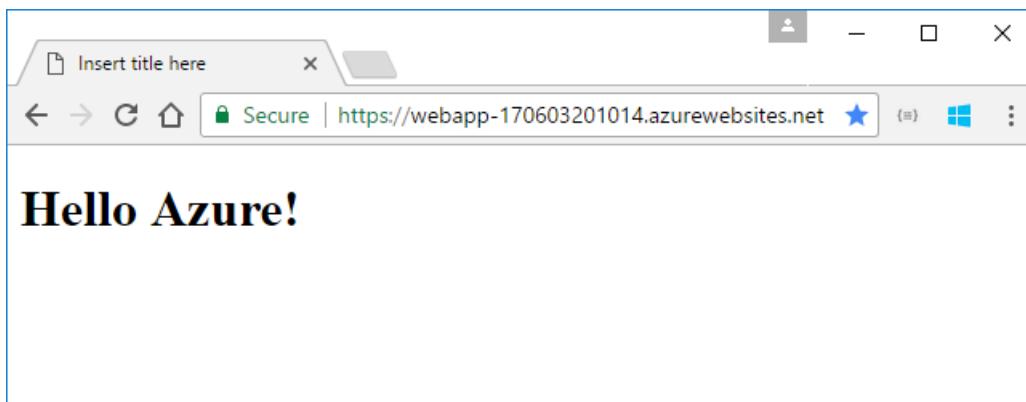
[PHP with MySQL](#)

Create your first Java web app in Azure

11/22/2017 • 4 min to read • [Edit Online](#)

Azure [Web Apps](#) provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy a Java web app to App Service by using the [Eclipse IDE for Java EE Developers](#).

When you have completed this quickstart, your application will look similar to the following illustration when you view it in a web browser:



Prerequisites

To complete this quickstart, install:

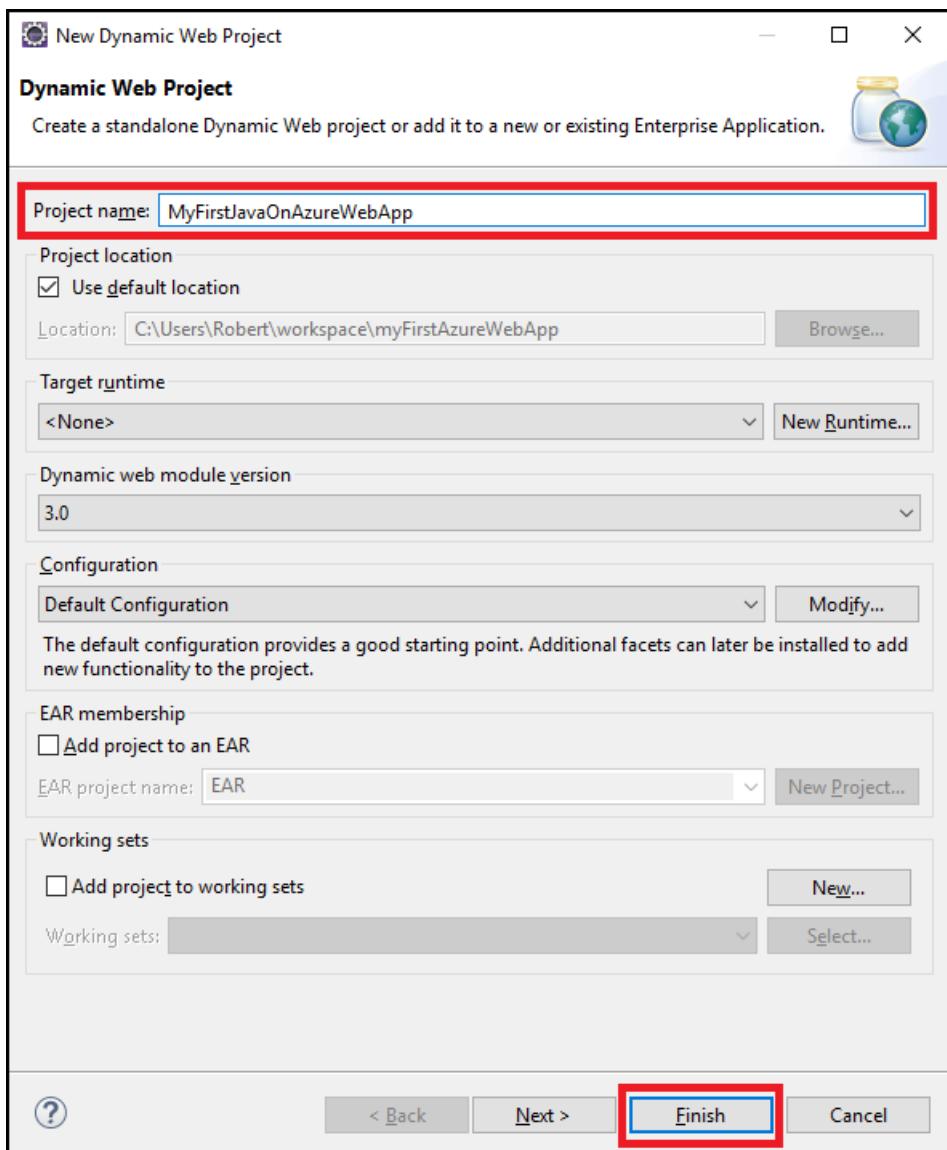
- The free [Eclipse IDE for Java EE Developers](#). This quickstart uses Eclipse Neon.
- The [Azure Toolkit for Eclipse](#).

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

Create a dynamic web project in Eclipse

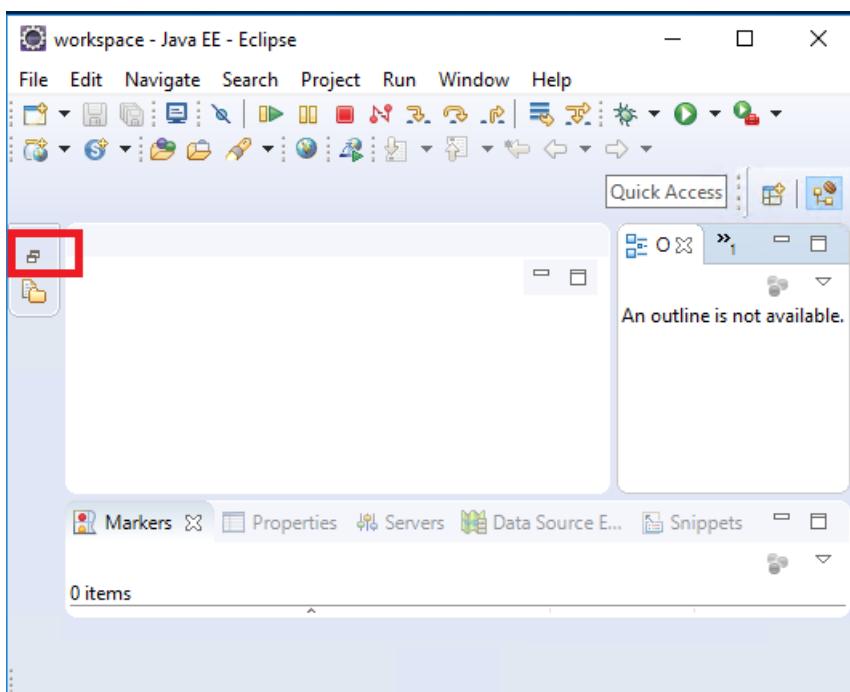
In Eclipse, select **File > New > Dynamic Web Project**.

In the **New Dynamic Web Project** dialog box, name the project **MyFirstJavaOnAzureWebApp**, and select **Finish**.

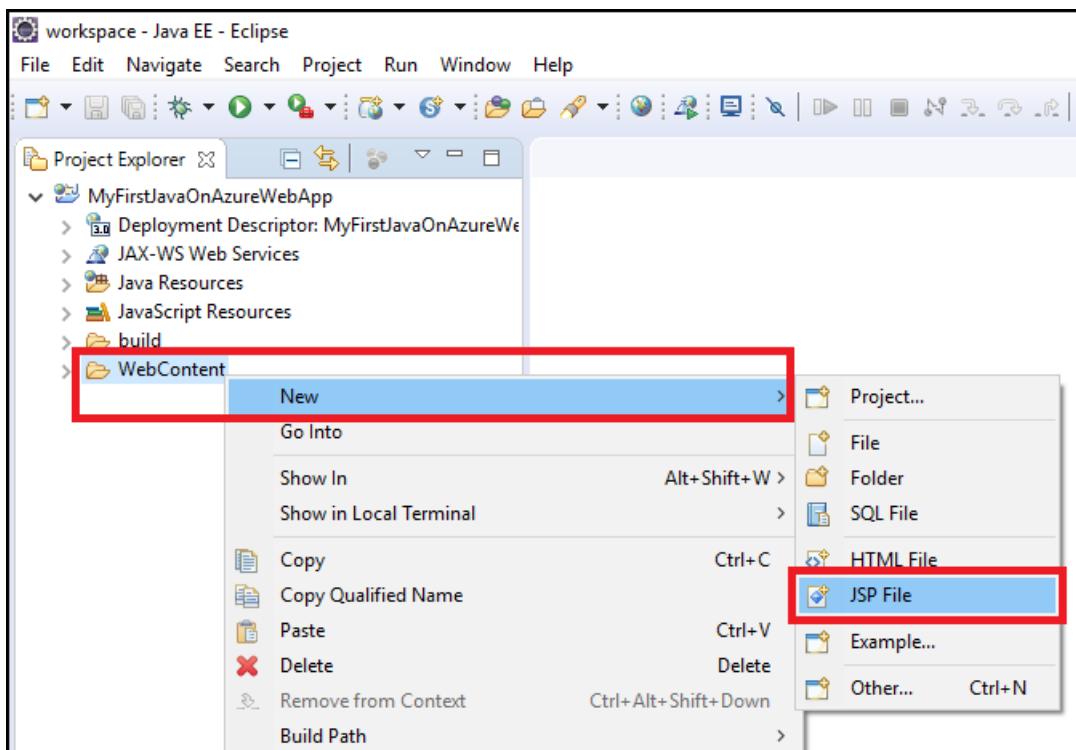


Add a JSP page

If Project Explorer is not displayed, restore it.

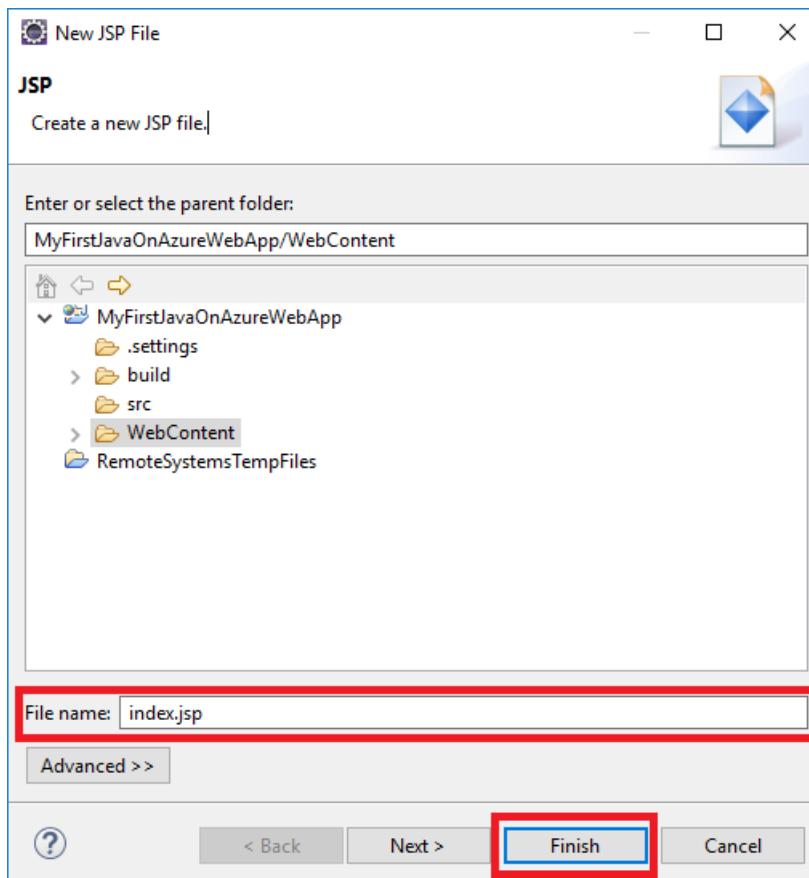


In Project Explorer, expand the **MyFirstJavaOnAzureWebApp** project. Right-click **WebContent**, and then select **New > JSP File**.



In the **New JSP File** dialog box:

- Name the file **index.jsp**.
- Select **Finish**.



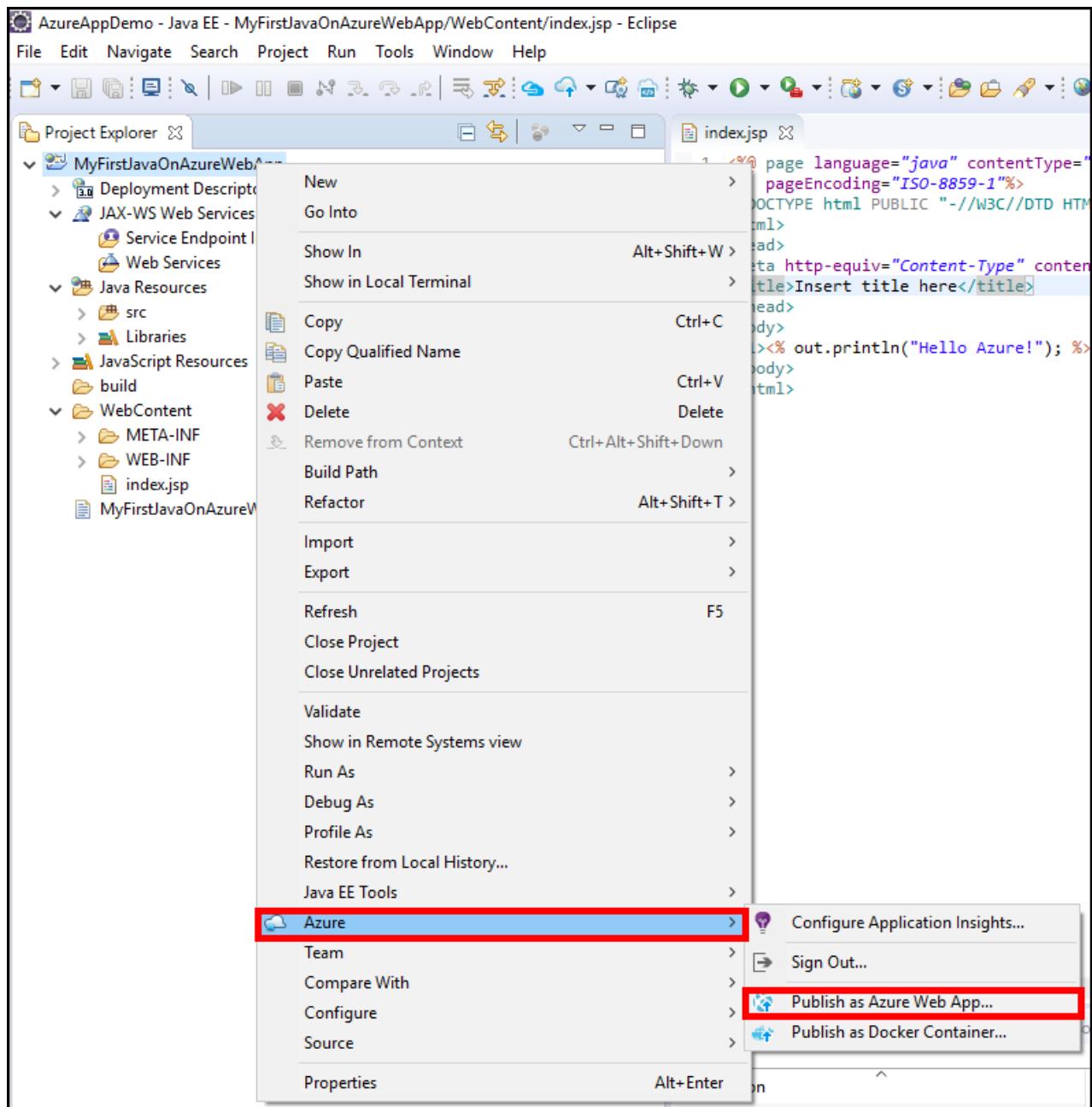
In the index.jsp file, replace the `<body></body>` element with the following markup:

```
<body>
<h1><% out.println("Hello Azure!"); %></h1>
</body>
```

Save the changes.

Publish the web app to Azure

In Project Explorer, right-click the project, and then select **Azure > Publish as Azure Web App**.



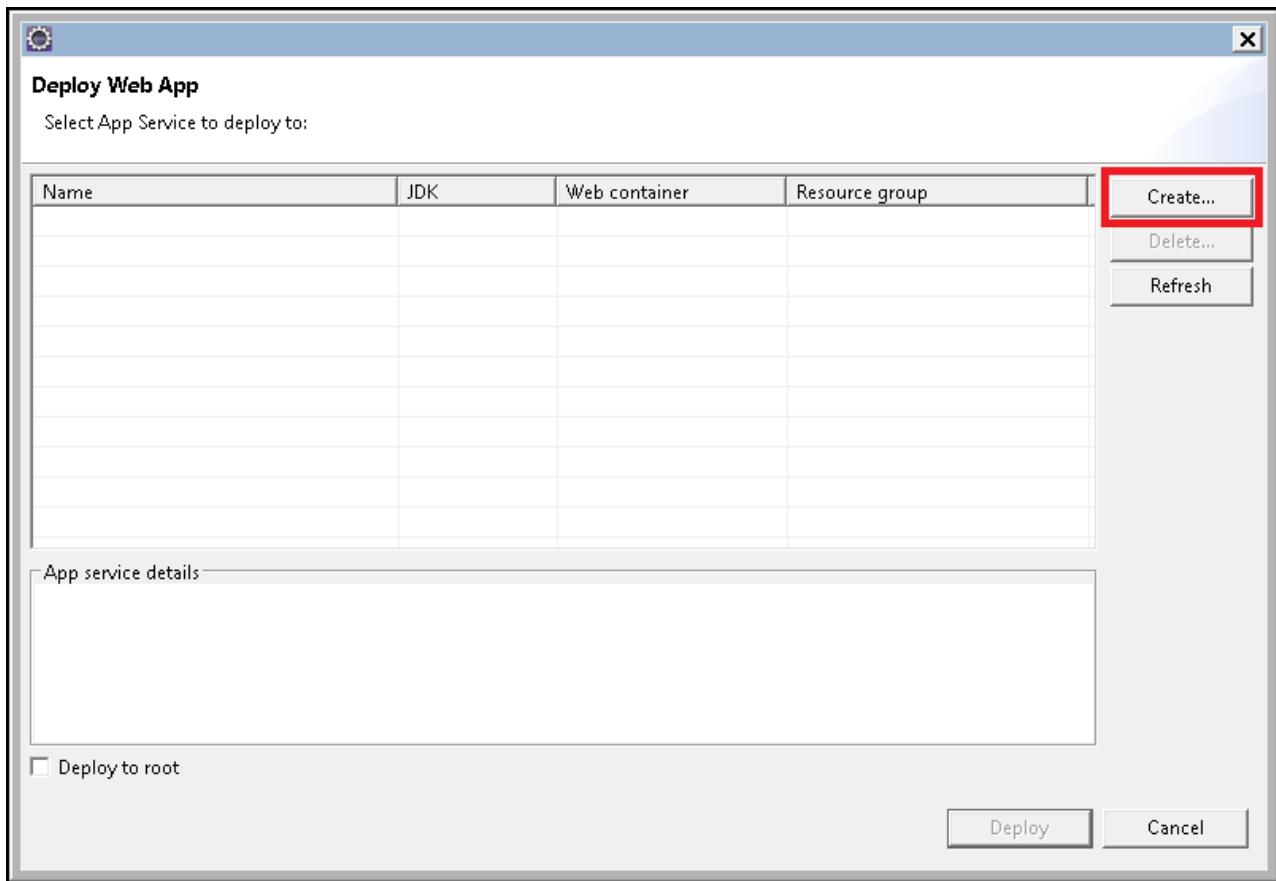
In the **Azure Sign In** dialog box, keep the **Interactive** option, and then select **Sign in**.

Follow the sign-in instructions.

Deploy Web App dialog box

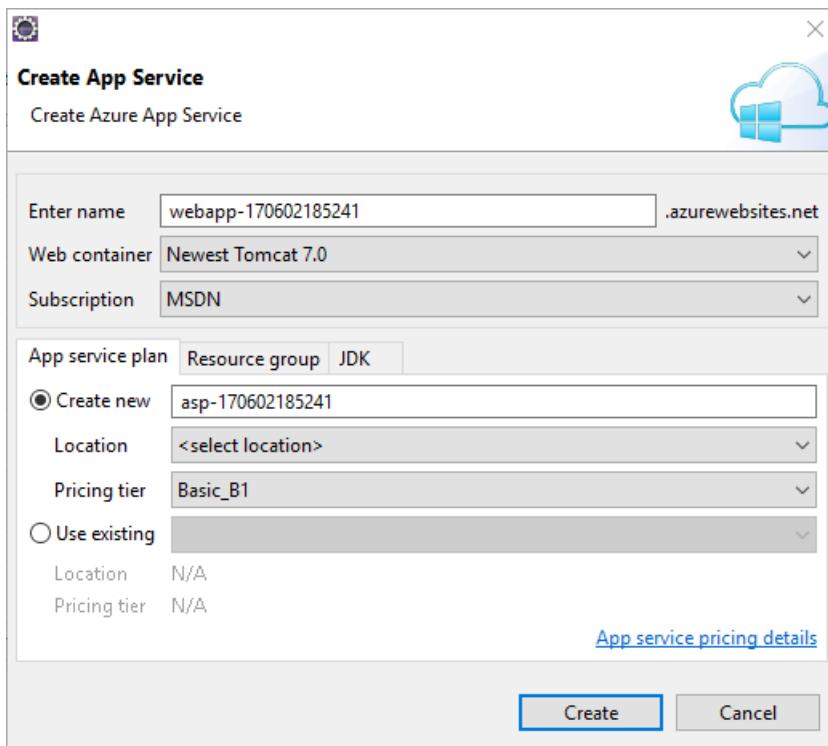
After you have signed in to your Azure account, the **Deploy Web App** dialog box appears.

Select **Create**.



Create App Service dialog box

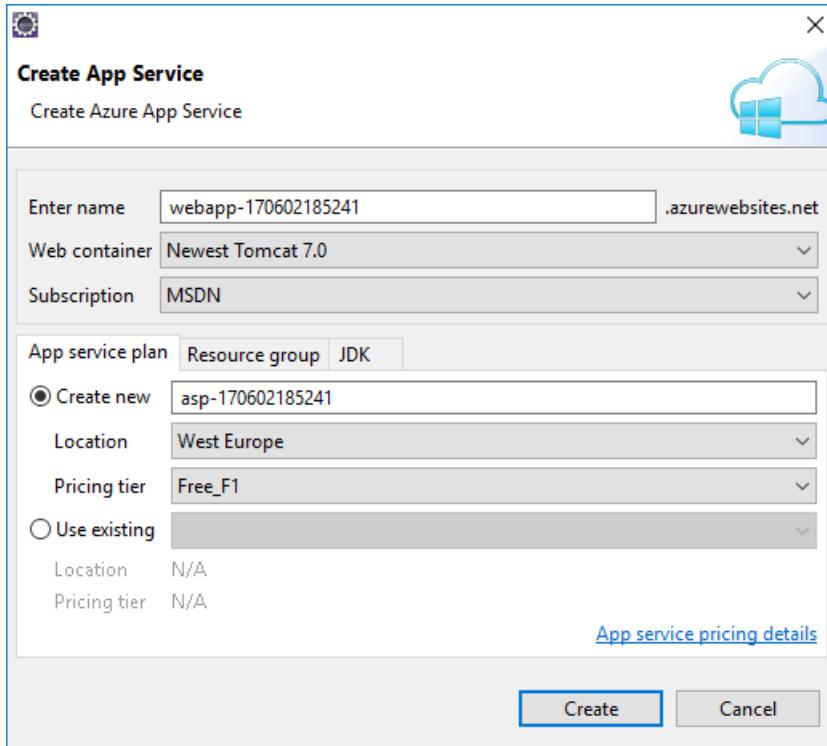
The **Create App Service** dialog box appears with default values. The number **170602185241** shown in the following image is different in your dialog box.



In the **Create App Service** dialog box:

- Keep the generated name for the web app. This name must be unique across Azure. The name is part of the URL address for the web app. For example: if the web app name is **MyJavaWebApp**, the URL is *myjavawebapp.azurewebsites.net*.
- Keep the default web container.

- Select an Azure subscription.
- On the **App service plan** tab:
 - **Create new:** Keep the default, which is the name of the App Service plan.
 - **Location:** Select **West Europe** or a location near you.
 - **Pricing tier:** Select the free option. For features, see [App Service pricing](#).



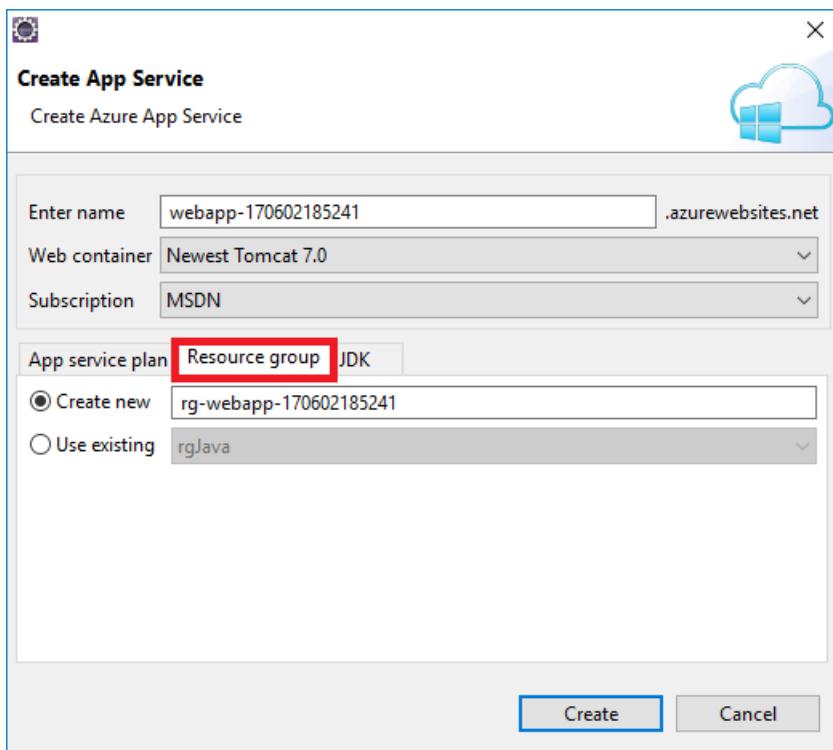
An [App Service plan](#) specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

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)

Resource group tab

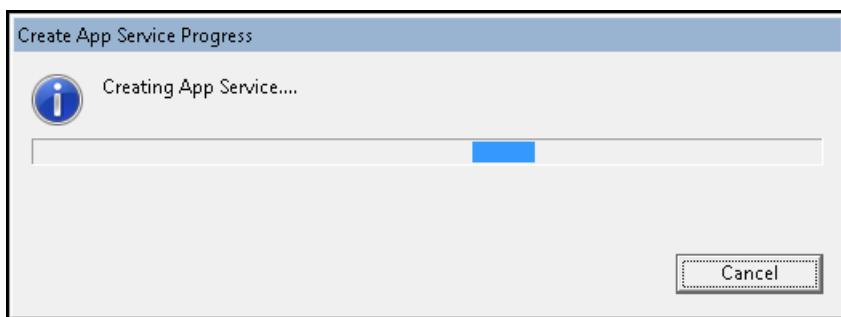
Select the **Resource group** tab. Keep the default generated value for the resource group.



A [resource group](#) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

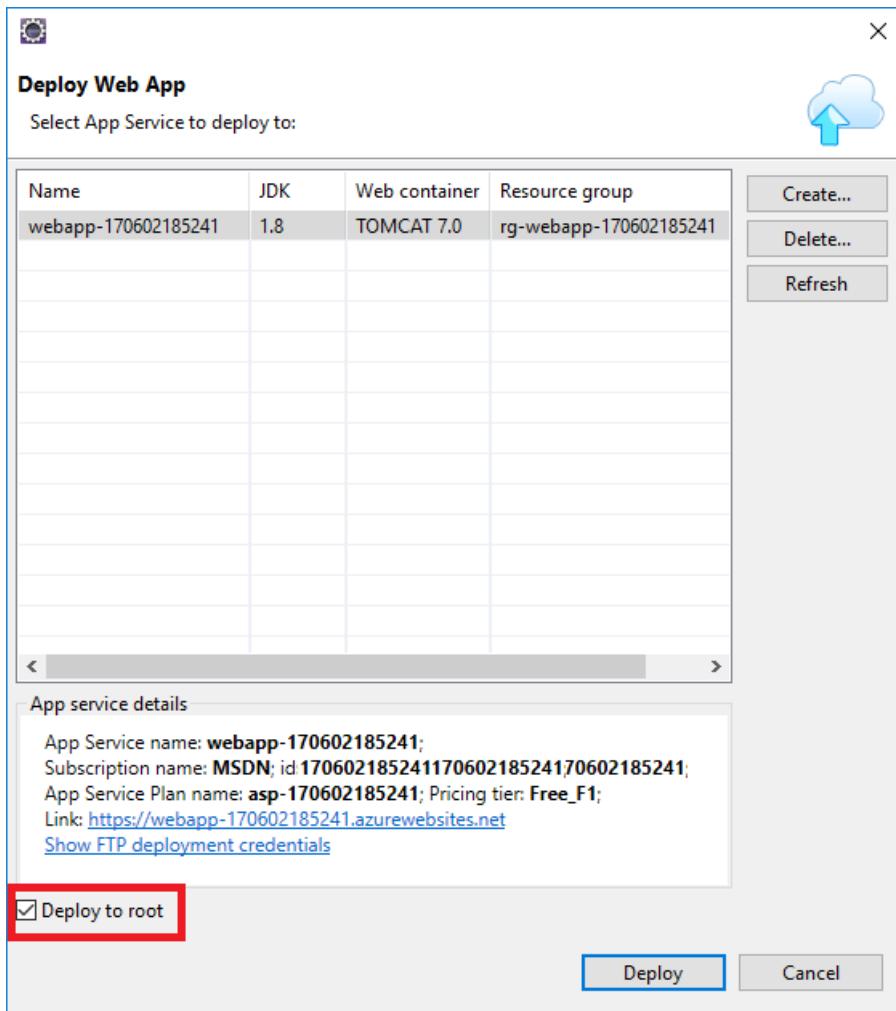
Select **Create**.

The Azure Toolkit creates the web app and displays a progress dialog box.



Deploy Web App dialog box

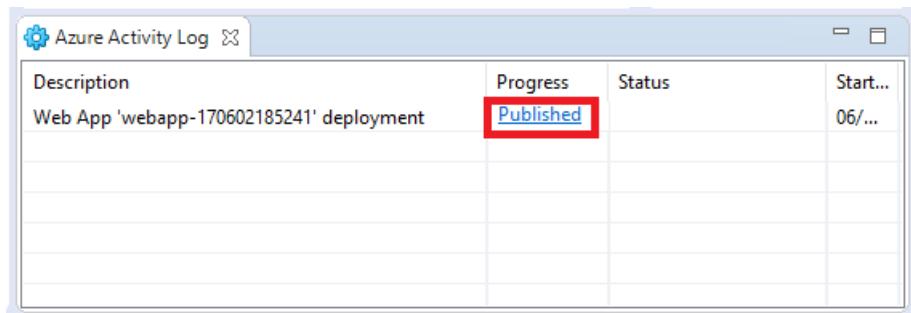
In the **Deploy Web App** dialog box, select **Deploy to root**. If you have an app service at wingtiptoyos.azurewebsites.net and you do not deploy to the root, the web app named **MyFirstJavaOnAzureWebApp** is deployed to wingtiptoyos.azurewebsites.net/MyFirstJavaOnAzureWebApp.



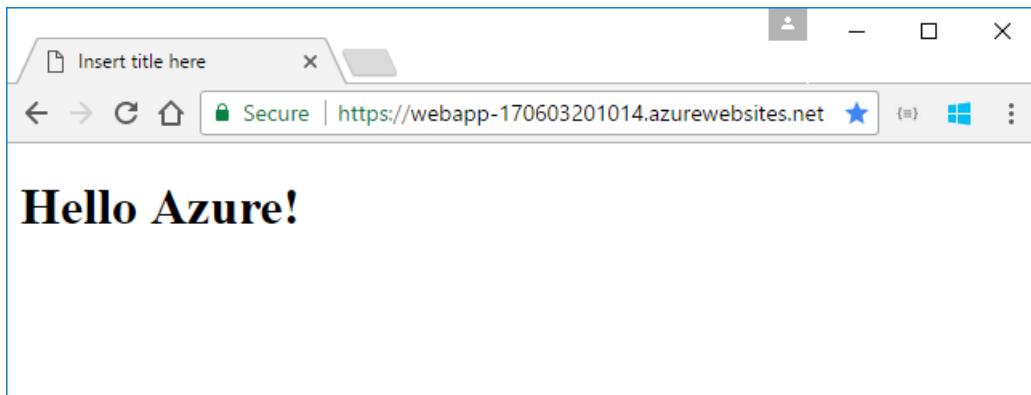
The dialog box shows the Azure, JDK, and web container selections.

Select **Deploy** to publish the web app to Azure.

When the publishing finishes, select the **Published** link in the **Azure Activity Log** dialog box.



Congratulations! You have successfully deployed your web app to Azure.



Update the web app

Change the sample JSP code to a different message.

```
<body>
<h1><% out.println("Hello again Azure!"); %></h1>
</body>
```

Save the changes.

In Project Explorer, right-click the project, and then select **Azure > Publish as Azure Web App**.

The **Deploy Web App** dialog box appears and shows the app service that you previously created.

NOTE

Select **Deploy to root** each time you publish.

Select the web app and select **Deploy**, which publishes the changes.

When the **Publishing** link appears, select it to browse to the web app and see the changes.

Manage the web app

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

From the left menu, select **Resource Groups**.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes 'Microsoft Azure', a search icon, a notifications icon (3 notifications), and user profile icons. Below the navigation is a dark blue header with the text 'Resource groups' and 'rickaOrgName'. A red box highlights the 'Resource groups' icon in the sidebar. The main content area displays a table with one item:

NAME	SUBSCRIPTION	LOCATION	...
rg-webapp-170602193915	MSDN	Central US	...

Select the resource group. The page shows the resources that you created in this quickstart.

The screenshot shows the Azure Resource Group Overview page for 'rg-webapp-170602193915'. On the left, there's a sidebar with various tabs: Overview, Activity log, Access control (IAM), Tags, Quickstart, Resource costs, Deployments, and Policies. The 'Overview' tab is selected. The main area displays two items in a table:

NAME	TYPE	LOCATION
asp-170602193915	App Service plan	Central US
webapp-170602193915	App Service	Central US

Select the web app (**webapp-170602193915** in the preceding image).

The **Overview** page appears. This page gives you a view of how the app is doing. Here, you can perform basic management tasks like browse, stop, start, restart, and delete. The tabs on the left side of the page show the different configurations that you can open.

The screenshot shows the Azure App Service Overview page for 'webapp-170602193915'. The left sidebar includes tabs for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment slots, Deployment options, Continuous Delivery (Preview), Application settings, Authentication / Authorization, Backups, Custom domains, and SSL certificates. The 'Overview' tab is selected. The main content area contains essential information and a monitoring chart.

Essentials

- Resource group ([change](#)) **rg-webapp-170602193915**
- Status **Running**
- Location **Central US**
- Subscription name ([change](#)) **MSDN**
- Subscription ID **74812a68-74812a68-74812a68-8422aec1**
- URL **<http://webapp-170602193915.azurewebsites.net>**
- App Service plan/pricing tier **asp-170602193915 (Basic: 1 Small)**
- FTP/deployment username **webapp-170602193915\rick@!#%**
- FTP hostname **<ftp://waws-prod-dm1-007.ftp.azurewebsites.net>**
- FTPS hostname **<https://waws-prod-dm1-007.ftp.azurewebsites.net>**

Monitoring

Requests and errors

Time	REQUESTS	HTTP SERVER ERRORS
8:15 AM	0	0
8:30 AM	0	0
8:45 AM	0	0
9 AM	30	0

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

[Map custom domain](#)

Create a Python web app in Azure

11/22/2017 • 7 min to read • [Edit Online](#)

[Azure Web Apps](#) provides a highly scalable, self-patching web hosting service. This quickstart walks through how to develop and deploy a Python app to Azure Web Apps. You create the web app using the [Azure CLI](#), and you use Git to deploy sample Python code to the web app.



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

Prerequisites

To complete this tutorial:

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

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

Download the sample

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

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

Change to the directory that contains the sample code.

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

Run the app locally

Install the required packages using `pip`.

```
pip install -r requirements.txt
```

Run the application locally by opening a terminal window and using the `Python` command to launch the built-in Python web server.

```
python main.py
```

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

You can 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.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	A screenshot of the Azure portal interface. A specific button labeled "Cloud Shell" is highlighted with a red box. This button is part of a horizontal menu bar in the top right corner of the portal. Other buttons visible include "Search", "Help", and "More".
Click the Cloud Shell button on the menu in the upper right of the Azure portal.	

Create a deployment user

In the Cloud Shell, create deployment credentials with the [az webapp deployment user set](#) command. A deployment user is required for FTP and local Git deployment to a web app. The user name and password are account level. *They are different from your Azure subscription credentials.*

In the following example, replace <username> and <password> (including brackets) with a new user name and password. The user name must be unique. The password must be at least eight characters long, with two of the following three elements: letters, numbers, symbols.

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

If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

You create this deployment user only once; you can use it for all your Azure deployments.

NOTE

Record the user name and password. You need them to deploy the web app later.

Create a resource group

In the Cloud Shell, create a resource group with the [az group create](#) command.

A [resource group](#) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

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

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

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

Create an Azure App Service plan

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

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)

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/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 [python|3.4](#). To see all supported runtimes, run [az webapp list-runtimes](#).

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime "python|3.4" --deployment-local-git
```

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

```
Local git is configured with url of 'https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git'  
{  
    "availabilityState": "Normal",  
    "clientAffinityEnabled": true,  
    "clientCertEnabled": false,  
    "cloningInfo": null,  
    "containerSize": 0,  
    "dailyMemoryTimeQuota": 0,  
    "defaultHostName": "<app_name>.azurewebsites.net",  
    "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.

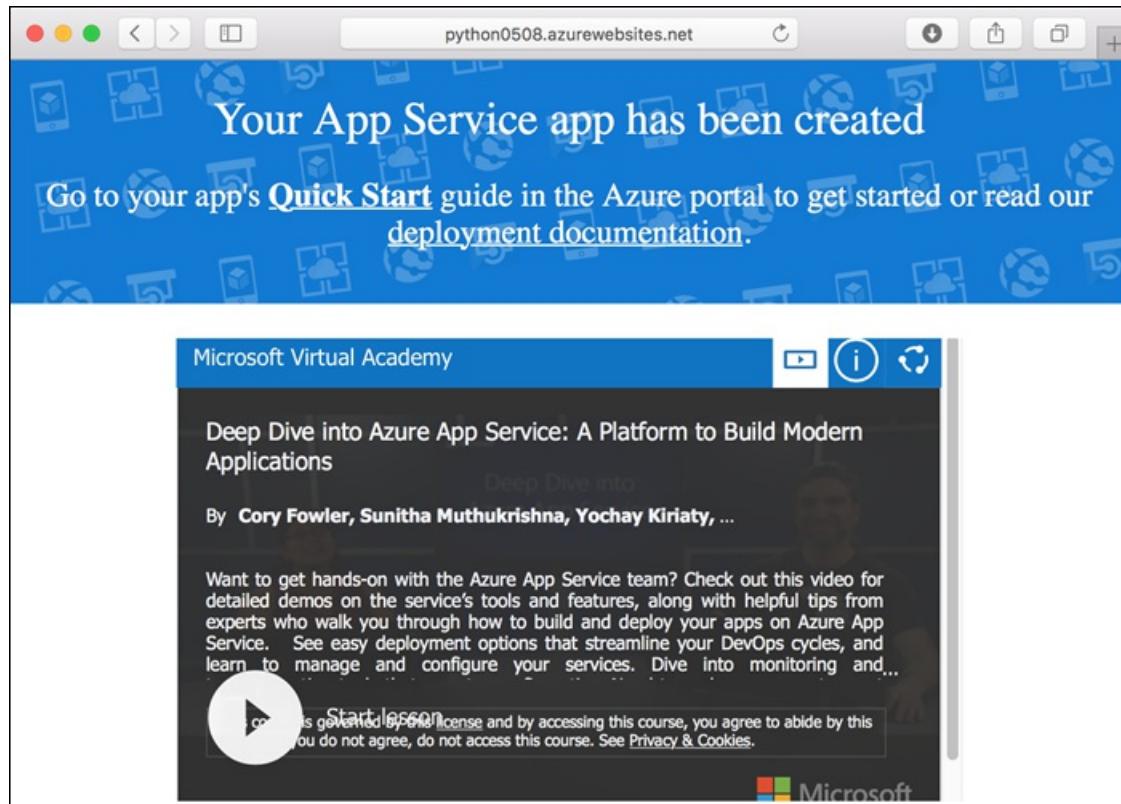
NOTE

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'll need it later.

Browse to your newly created web app. Replace `<app name>` with a unique app name.

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



Push to Azure from Git

In the local terminal window, add an Azure remote to your local Git repository. This Azure remote was created for you in [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 prompted for a password, make sure that you enter the password you created in [Configure a deployment user](#), not the password you use to log 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: 18, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (18/18), 4.31 KiB | 0 bytes/s, done.
Total 18 (delta 4), reused 0 (delta 0)
remote: Updating branch 'master'.
remote: Updating submodules.
remote: Preparing deployment for commit id '44e74fe7dd'.
remote: Generating deployment script.
remote: Generating deployment script for python Web Site
remote: Generated deployment script files
remote: Running deployment command...
remote: Handling python deployment.
remote: KuduSync.NET from: 'D:\home\site\repository' to: 'D:\home\site\wwwroot'
remote: Deleting file: 'hostingstart.html'
remote: Copying file: '.gitignore'
remote: Copying file: 'LICENSE'
remote: Copying file: 'main.py'
remote: Copying file: 'README.md'
remote: Copying file: 'requirements.txt'
remote: Copying file: 'virtualenv_proxy.py'
remote: Copying file: 'web.2.7.config'
remote: Copying file: 'web.3.4.config'
remote: Detected requirements.txt. You can skip Python specific steps with a .skipPythonDeployment file.
remote: Detecting Python runtime from site configuration
remote: Detected python-3.4
remote: Creating python-3.4 virtual environment.
remote: .....
remote: Pip install requirements.
remote: Successfully installed Flask click itsdangerous Jinja2 Werkzeug MarkupSafe
remote: Cleaning up...
remote: .
remote: Overwriting web.config with web.3.4.config
remote:      1 file(s) copied.
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Deployment successful.
To https://<app_name>.scm.azurewebsites.net/<app_name>.git
 * [new branch]      master -> master
```

Browse to the app

Browse to the deployed application using your web browser.

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

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



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

Update and redeploy the code

Using a local text editor, open the `main.py` file in the Python app, and make a small change to the text next to the `return` statement:

```
return '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, switch back to the browser window that opened in the [Browse to the app](#) step, and refresh the page.



Manage your new Azure web app

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

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

Subscriptions: 1 of 19 selected – Don't see a subscription? [Switch directories](#)

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

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

nodejs-docs-hello-world

Overview

Browse Stop Swap Restart Delete More

Essentials

Resource group ([change](#))
myResourceGroup

Status
Running

Location
West Europe

Subscription name ([change](#))
Field Engineer Demo

Subscription ID

URL
[http://nodejs-docs-hello-world.azurewebsites...](http://nodejs-docs-hello-world.azurewebsites.net)

App Service plan/pricing tier
quickStartPlan (Standard: 1 Small)

FTP/deployment username
nodejs-docs-hello-world\demoaccount

FTP hostname
<ftp://waws-prod-am2-085.ftp.azurewebsite...>

FTPS hostname
<https://waws-prod-am2-085.ftp.azurewebsite...>

Monitoring

Requests and errors

The left menu provides different pages for configuring your app.

Clean up resources

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

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

This command may take a minute to run.

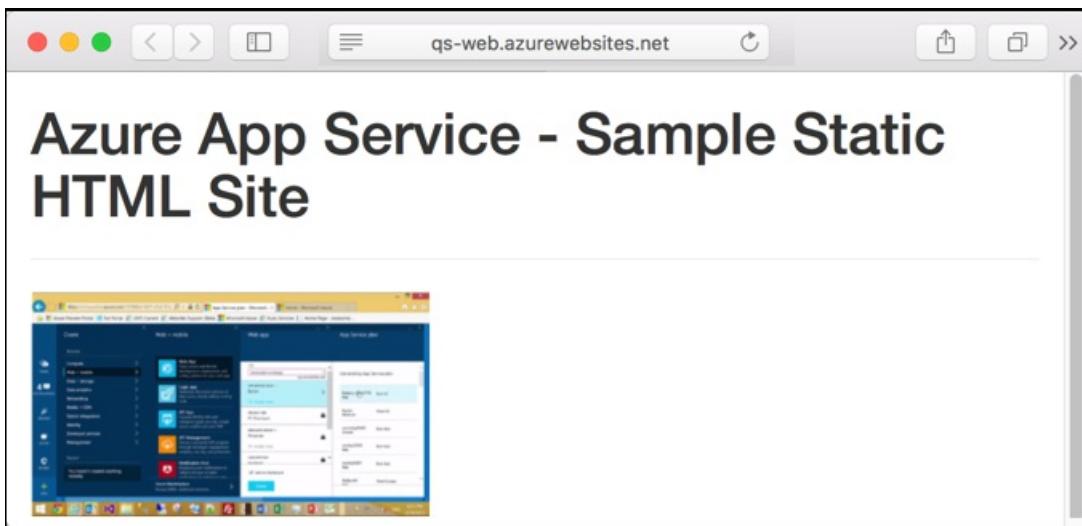
Next steps

[Map an existing custom DNS name to Azure Web Apps](#)

Create a static HTML web app in Azure

11/22/2017 • 6 min to read • [Edit Online](#)

Azure Web Apps provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy a basic HTML+CSS site to Azure Web Apps. You create the web app using the [Azure CLI](#), and you use Git to deploy sample HTML content to the web app.



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

Prerequisites

To complete this quickstart:

- [Install Git](#)

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

Download the sample

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

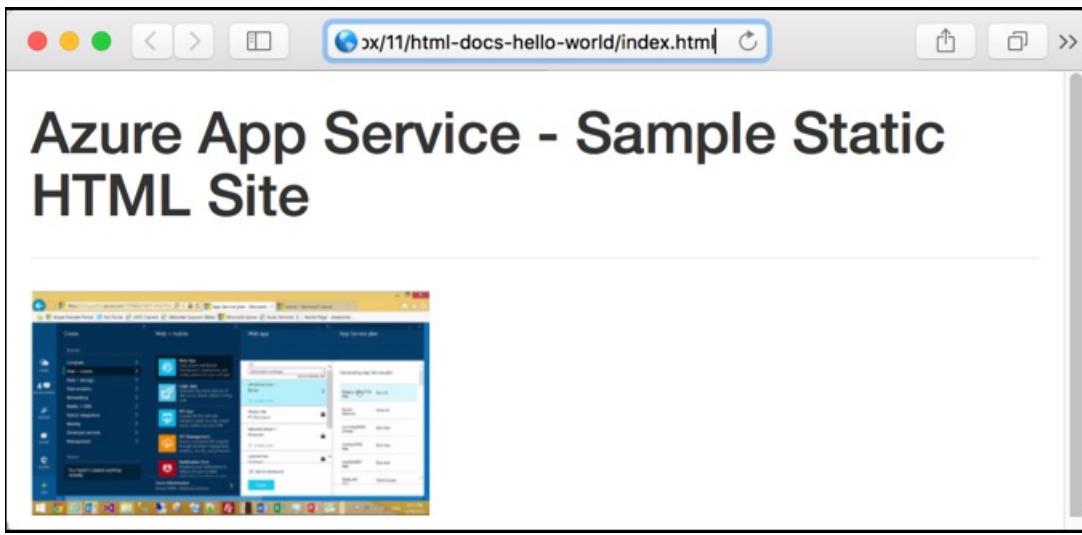
```
git clone https://github.com/Azure-Samples/html-docs-hello-world.git
```

Change to the directory that contains the sample code.

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

View the HTML

Navigate to the directory that contains the sample HTML. Open the *index.html* file in your browser.



Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	

Create a deployment user

In the Cloud Shell, create deployment credentials with the [az webapp deployment user set](#) command. A deployment user is required for FTP and local Git deployment to a web app. The user name and password are account level. *They are different from your Azure subscription credentials.*

In the following example, replace <username> and <password> (including brackets) with a new user name and password. The user name must be unique. The password must be at least eight characters long, with two of the following three elements: letters, numbers, symbols.

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

If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

You create this deployment user only once; you can use it for all your Azure deployments.

NOTE

Record the user name and password. You need them to deploy the web app later.

Create a resource group

In the Cloud Shell, create a resource group with the [az group create](#) command.

A [resource group](#) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group named `myResourceGroup` in the *West Europe* location. To see all supported locations for App Service, run the `az appservice list-locations` command.

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

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

Create an Azure App Service plan

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

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)

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

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 `-`).

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

When the web app has been created, the Azure CLI shows information 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.

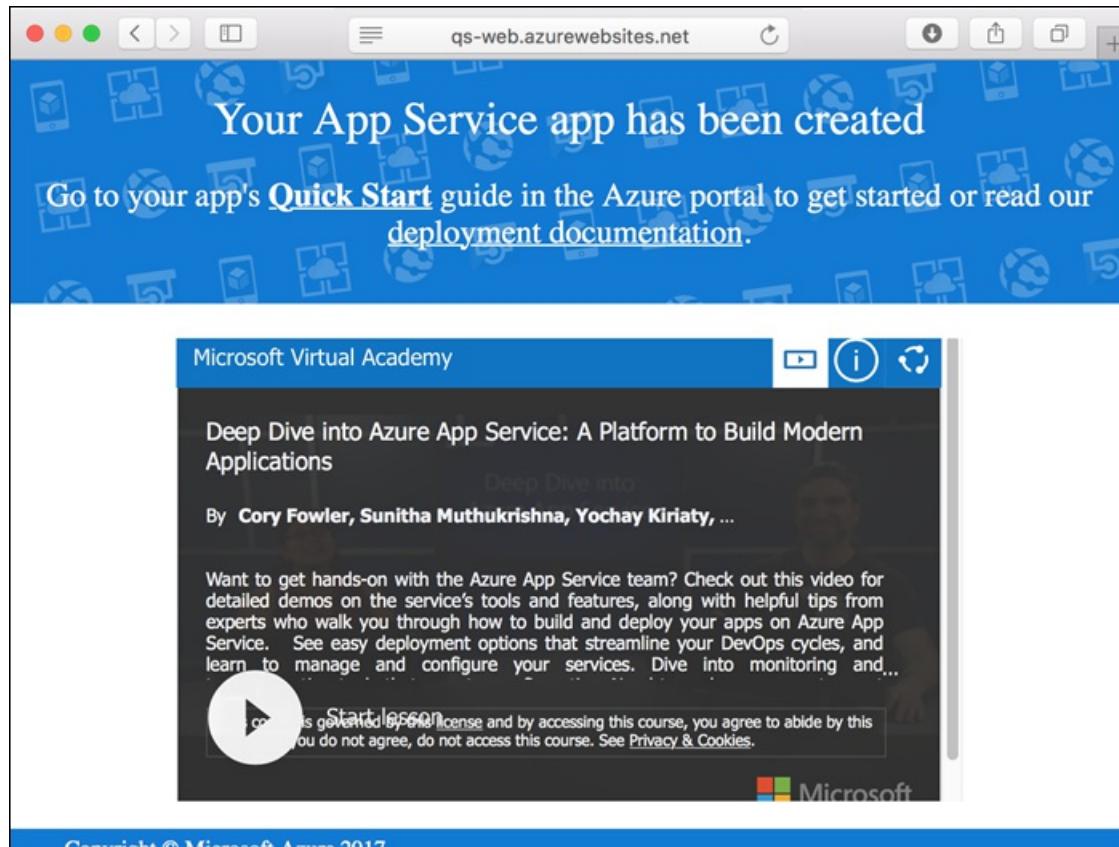
NOTE

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'll need it later.

Browse to the newly created web app.

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



Push to Azure from Git

In the local terminal window, add an Azure remote to your local Git repository. This Azure remote was created for you in [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 prompted for a password, make sure that you enter the password you created in [Configure a deployment user](#), not the password you use to log in to the Azure portal.

```
git push azure master
```

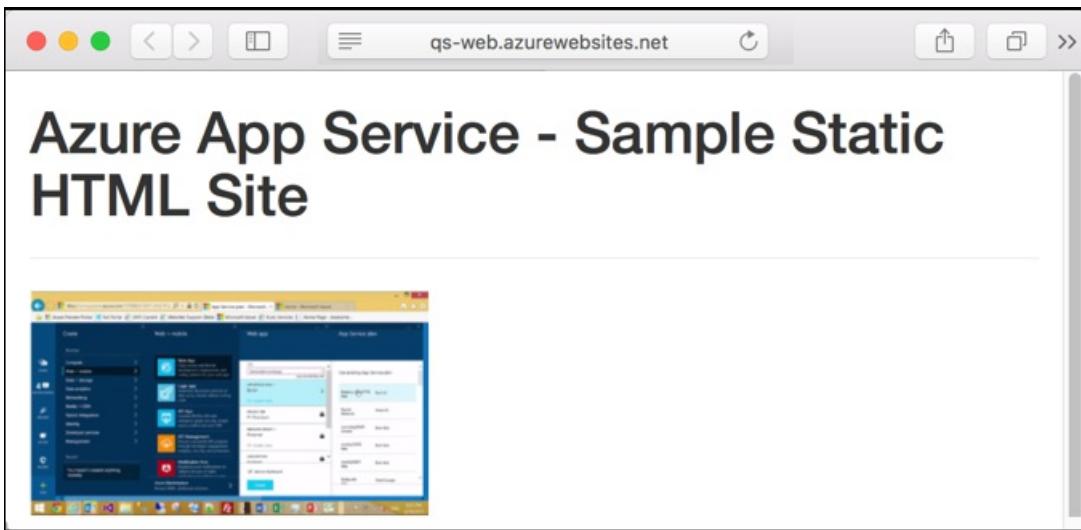
This command may take a few minutes to run. While running, it displays information similar to the following example:

```
Counting objects: 13, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (11/11), done.
Writing objects: 100% (13/13), 2.07 KiB | 0 bytes/s, done.
Total 13 (delta 2), reused 0 (delta 0)
remote: Updating branch 'master'.
remote: Updating submodules.
remote: Preparing deployment for commit id 'cc39b1e4cb'.
remote: Generating deployment script.
remote: Generating deployment script for Web Site
remote: Generated deployment script files
remote: Running deployment command...
remote: Handling Basic Web Site deployment.
remote: KuduSync.NET from: 'D:\home\site\repository' to: 'D:\home\site\wwwroot'
remote: Deleting file: 'hostingstart.html'
remote: Copying file: '.gitignore'
remote: Copying file: 'LICENSE'
remote: Copying file: 'README.md'
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Deployment successful.
To https://<app_name>.scm.azurewebsites.net/<app_name>.git
 * [new branch]      master -> master
```

Browse to the app

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

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



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

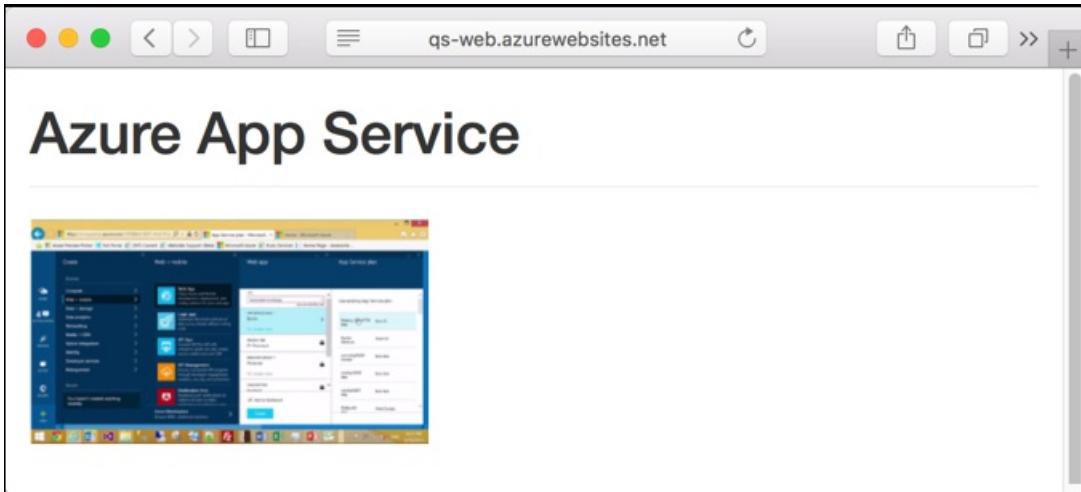
Update and redeploy the app

Open the *index.html* file in a text editor, and make a change to the markup. For example, change the H1 heading from "Azure App Service - Sample Static HTML Site" to just "Azure App Service".

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

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

Once deployment has completed, refresh your browser to see the changes.



Manage your new Azure web app

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

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

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Azure logo, 'Microsoft Azure', 'App Services', a search icon, a bell icon, a gear icon, a smiley face icon, a question mark icon, the user name 'ricka', and the organization name 'RICKAORGNAME'. Below the navigation bar is a header for 'App Services' with a 'Subscriptions' dropdown set to 'MSDN'. A 'Filter by name...' input field is present. The main content area displays a table with one item: 'qs-web'. The columns are 'NAME', 'RESOURCE...', 'STATUS', 'APP TYPE', and 'APP SERVICE PL...'. The 'qs-web' row shows 'qs-web' under 'NAME', 'myResourceG...' under 'RESOURCE...', 'Running' under 'STATUS', 'Web app' under 'APP TYPE', and 'quickStartPlan' under 'APP SERVICE PL...'. To the left of the main content is a sidebar with various icons: a plus sign, a magnifying glass, a gear, a question mark, a plus sign with a circle, a cloud, a cube, a clock, a globe, a database, a monitor, and a computer screen. The 'globe' icon is highlighted with a red box.

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

The screenshot shows the Microsoft Azure portal interface for the 'qs-web' app service. The top navigation bar and user information are identical to the previous screenshot. The main content area has a header 'qs-web' and a search bar. On the left is a navigation menu with 'Overview' selected. Other options include 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'DEPLOYMENT', and 'Quickstart'. The right pane is divided into sections: 'Essentials' (Resource group: myResourceGroup, Status: Running, Location: West Europe, Subscription name: MSDN, Subscription ID: 95cb-a68-95cb-4d59-95cb-0195cb-2aec1, URL: http://qs-web.azurewebsites.net, App Service plan/pricing: quickStartPlan (Free), Git/Deployment username: ricka, Git clone url: https://ricka@qs-web:443, FTP hostname: ftp://waws-prod-am), 'Monitoring' (Requests and errors: 100), and 'Requests and errors' (100). The 'Overview' tab is highlighted with a blue background.

The left menu provides different pages for configuring your app.

Clean up resources

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

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

This command may take a minute to run.

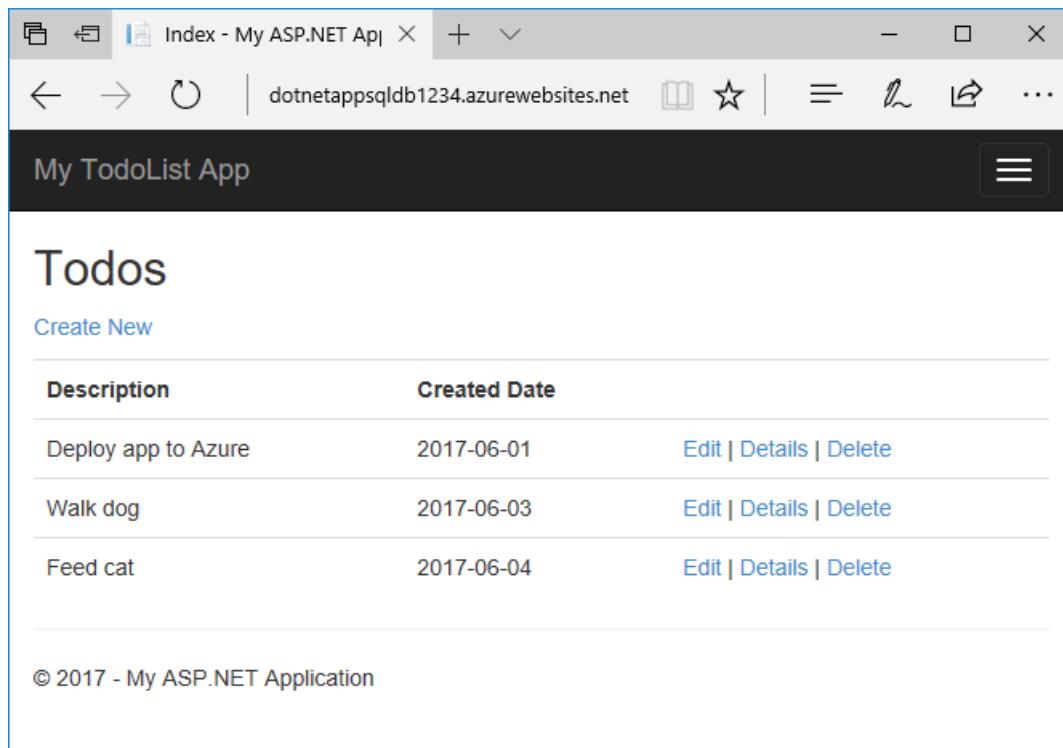
Next steps

[Map custom domain](#)

Build an ASP.NET app in Azure with SQL Database

10/26/2017 • 11 min to read • [Edit Online](#)

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

Prerequisites

To complete this tutorial:

- Install [Visual Studio 2017](#) with the following workloads:
 - **ASP.NET and web development**
 - **Azure development**

Installing — Visual Studio Enterprise 2017 — 15.2 (26430.12)

Workloads Individual components Language packs

Windows (3)

- Universal Windows Platform development
- .NET desktop development
- Desktop development with C++

Web & Cloud (7)

- ASP.NET and web development
- Azure development
- Python development
- Node.js development

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

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](#) CRUD (create-read-update-delete) 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.

Index - My ASP.NET App

localhost:1234

My TodoList App

Todos

Create New

Description	Created Date	
Walk dog	2017-06-01	Edit Details Delete
Build Azure ASP.NET app	2017-06-04	Edit Details Delete

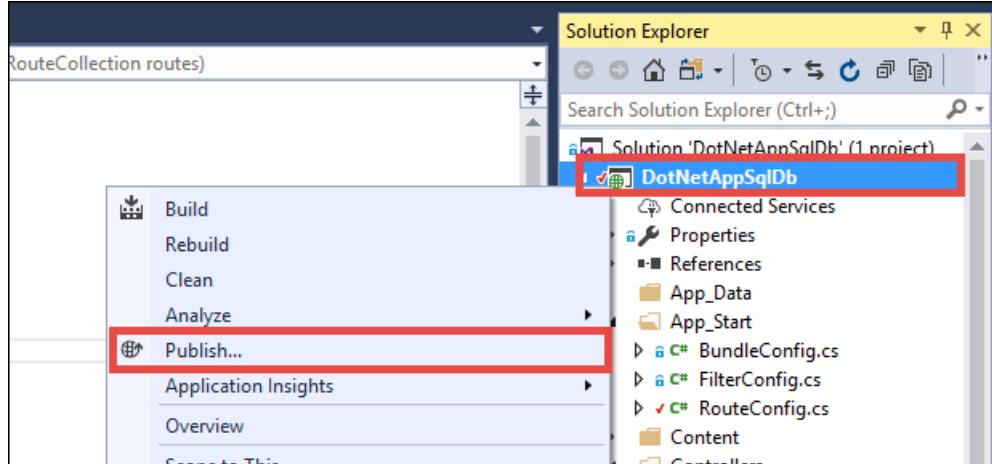
© 2017 - My ASP.NET Application

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

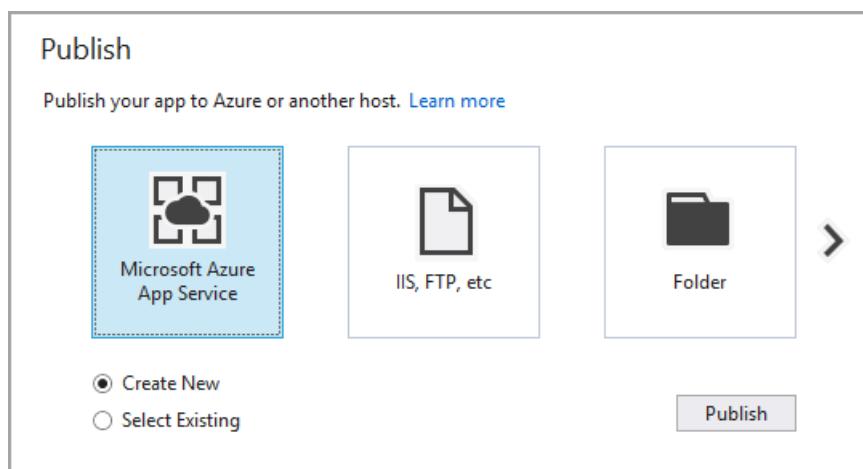
The app uses a database context to connect with the database. In this sample, the database context uses a connection string named `MyDbConnection`. The connection string is set in the `Web.config` file and referenced in the `Models/MyDatabaseContext.cs` file. The connection string name is used later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

Create App Service

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

Hosting **Services**

Web App Name: Change Type▼

Subscription:

Resource Group: New...

App Service Plan: New...

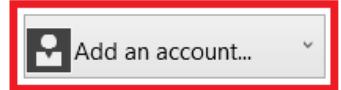
Clicking the Create button will create the following Azure resources

[Explore additional Azure services](#)

App Service - DotNetAppSqlDb1234

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
[Learn More](#)

Export... **Create** **Cancel**



Once signed in, you're ready to create all the resources you need for your Azure web app in this dialog.

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

Microsoft account

Hosting → **Services**

Web App Name: DotNetAppSqlDb1234 **Change Type** ▾

Subscription: MSDN

Resource Group: **New...**

App Service Plan: DotNetAppSqlDb20170608065141Plan* **New...**

Clicking the Create button will create the following Azure resources

[Explore additional Azure services](#)

App Service - DotNetAppSqlDb1234
App Service Plan - DotNetAppSqlDb20170608065141Plan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
[Learn More](#)

Export... **Create** (disabled) **Cancel**

NOTE

Do not click **Create**. You first need to set up a SQL Database in a later step.

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.

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

Create App Service

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

Microsoft account

Hosting

Web App Name: DotNetAppSqlDb1234

Change Type▼

Services

Subscription: MSDN

Resource Group: [New...](#)

App Service Plan: DotNetAppSqlDb1234Plan*

New...

Clicking the Create button will create the following Azure resources

[Explore additional Azure services](#)

App Service - DotNetAppSqlDb1234

App Service Plan - DotNetAppSqlDb1234Plan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.

[Learn More](#)

Export...  Cancel

Name the resource group **myResourceGroup**.

Create an App Service plan

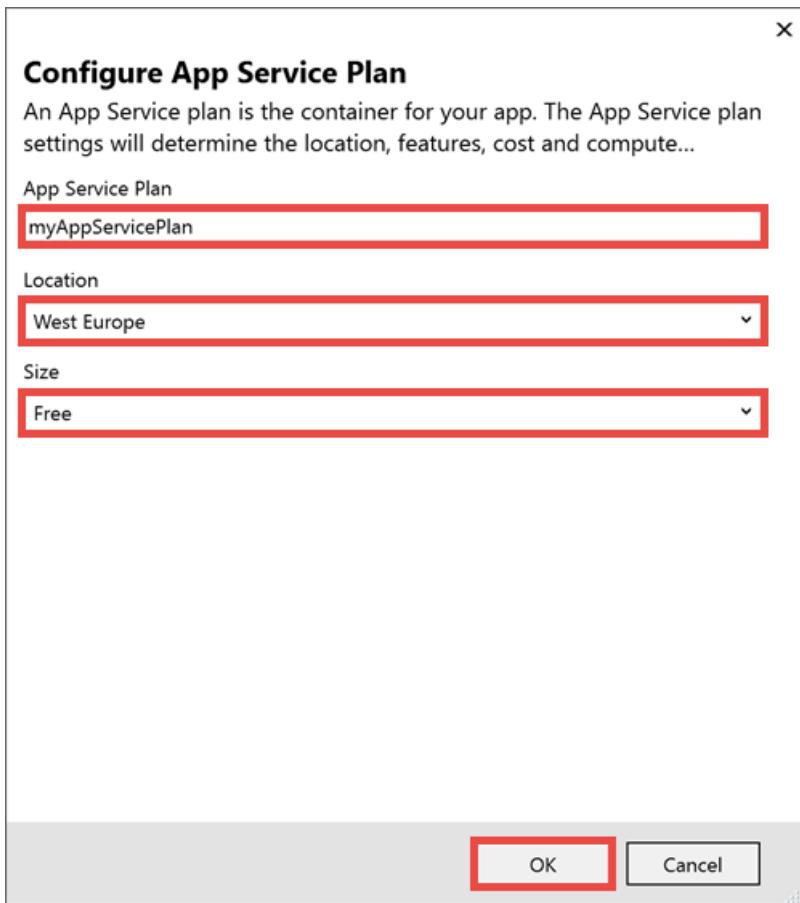
An [App Service plan](#) specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

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.

Select **Explore additional Azure services**.

Create App Service

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

Microsoft account

Hosting (i)

Services

Web App Name: DotNetAppSqlDb1234 Change Type ▾

Subscription: MSDN ▼

Resource Group: myResourceGroup* New... (i)

App Service Plan: myAppServicePlan* New... (i)

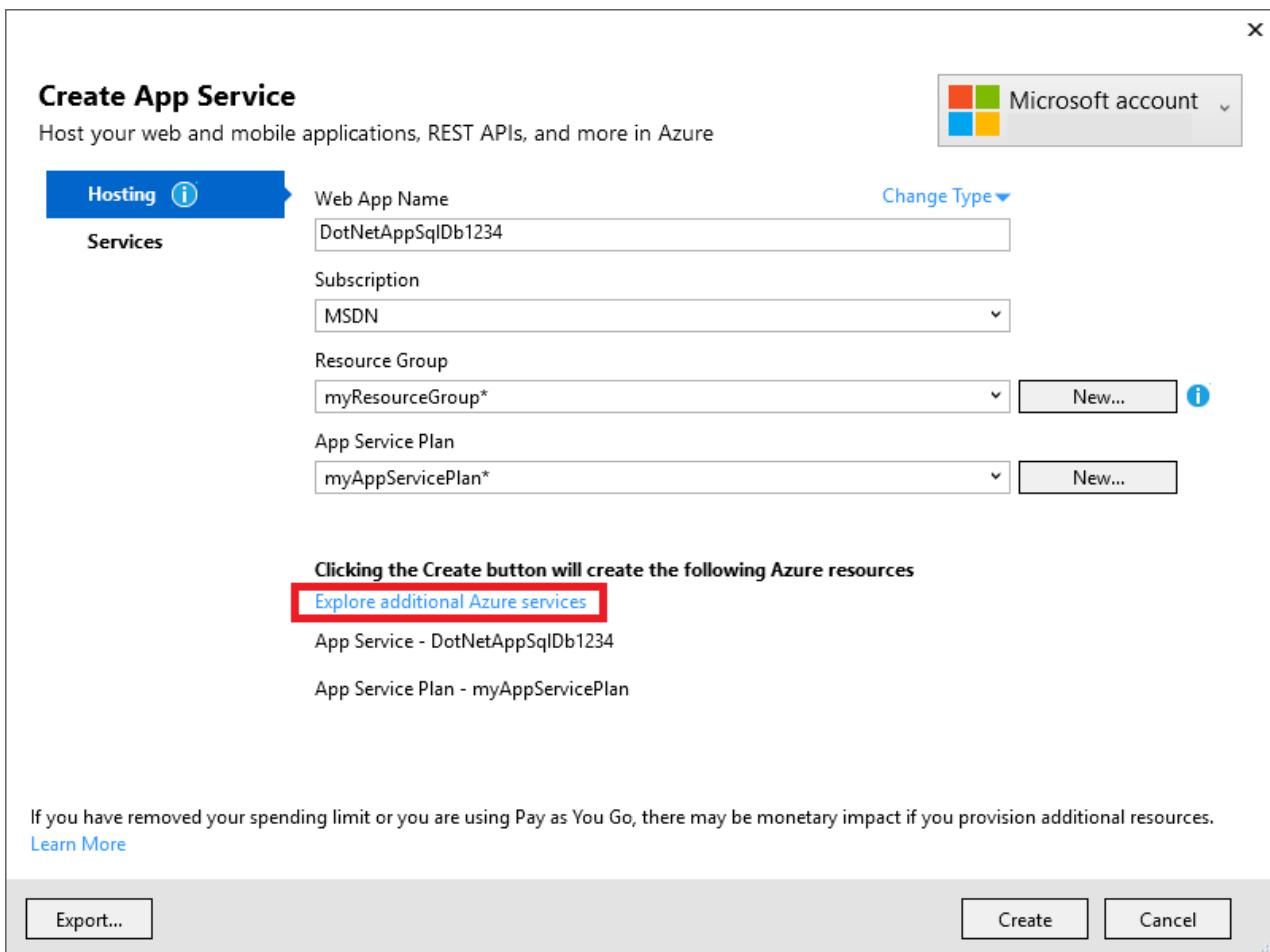
Clicking the Create button will create the following Azure resources

[Explore additional Azure services](#)

App Service - DotNetAppSqlDb1234
App Service Plan - myAppServicePlan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
[Learn More](#)

Export... Create (i) Cancel



In the **Services** tab, click the + icon next to **SQL Database**.

Create App Service

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

Microsoft account

Hosting (i)

Services

Select any additional Azure resources your app will need Show: Recommended ▾

Resource Type

 SQL Database
Scalable and managed database service for modern business-class apps + (i)

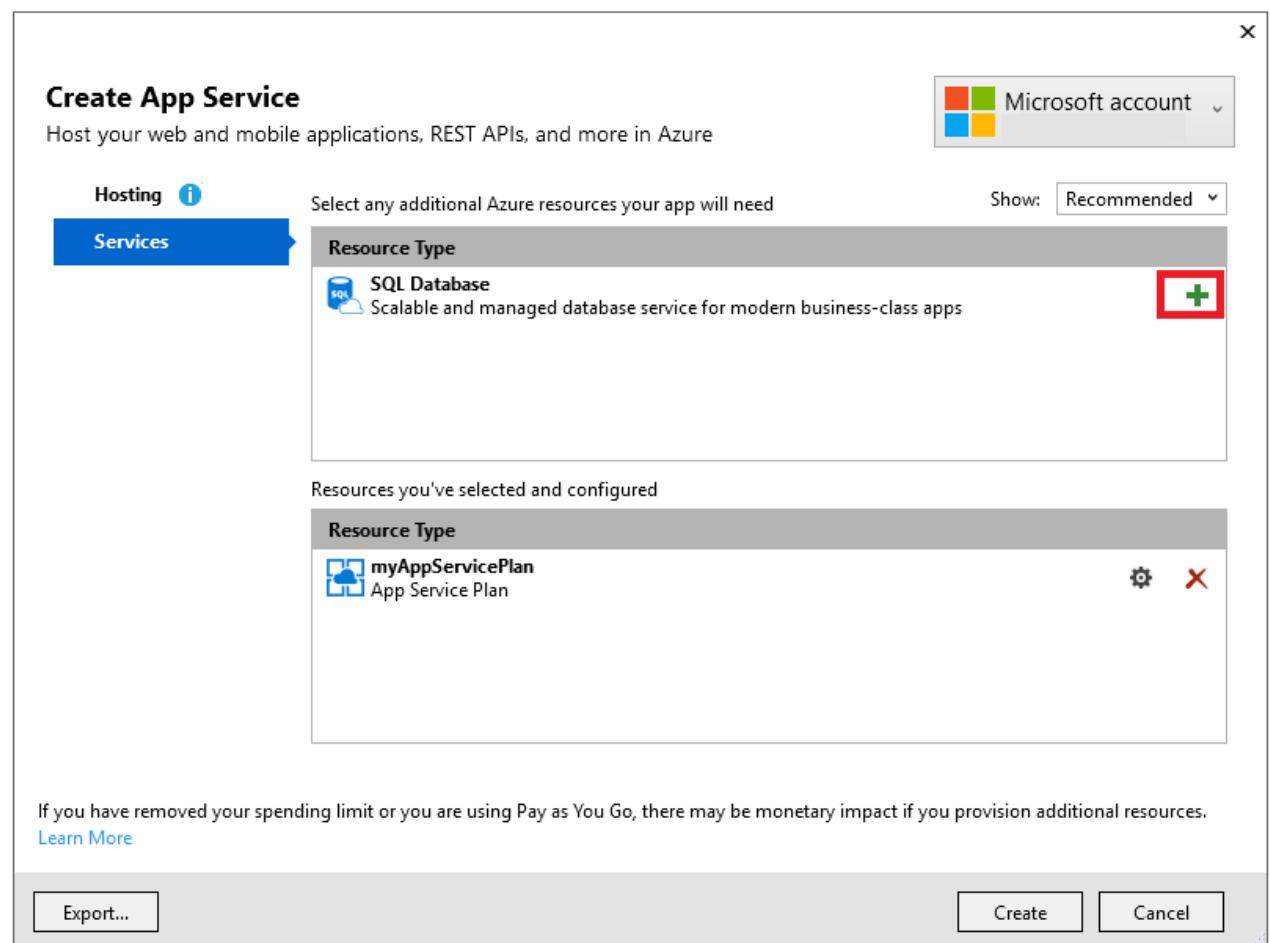
Resources you've selected and configured

Resource Type

 myAppServicePlan
App Service Plan ⚙️ (i) ✖️

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
[Learn More](#)

Export... Create (i) Cancel

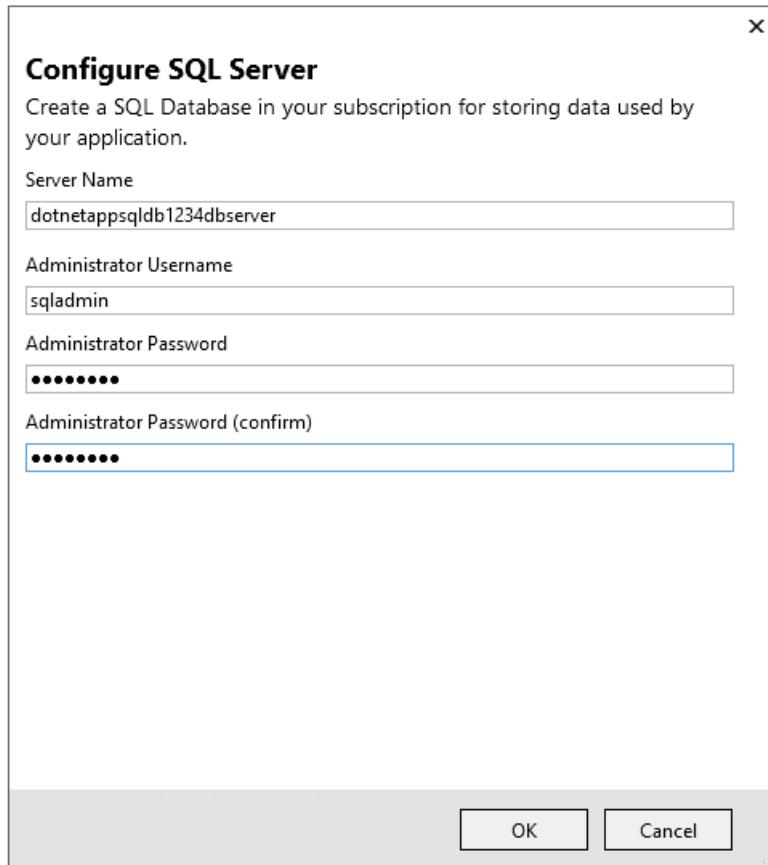


In the **Configure SQL Database** dialog, click **New** next to **SQL Server**.

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.

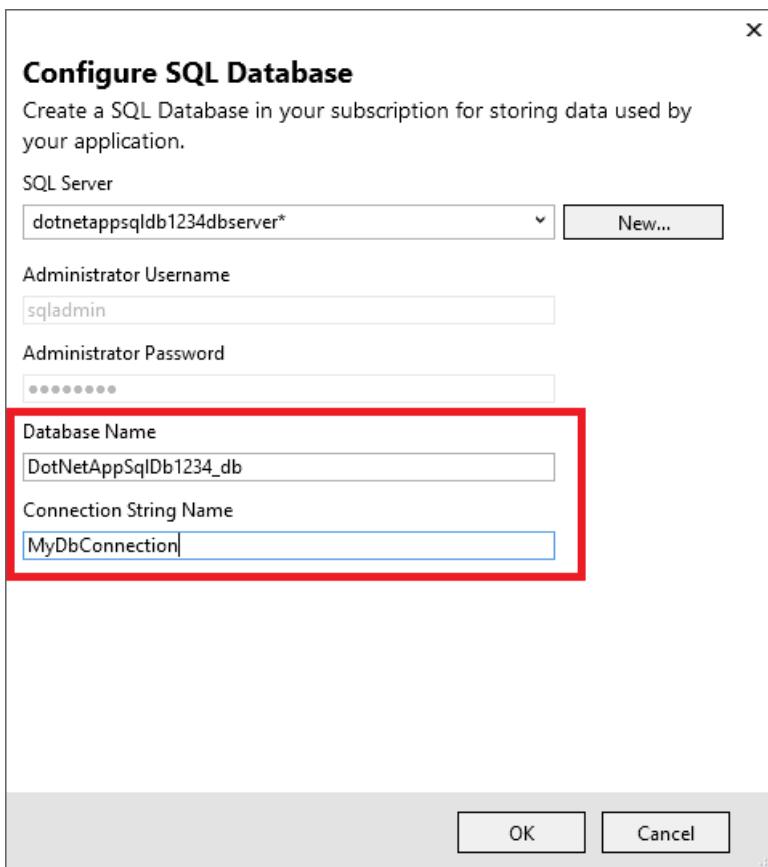


Click **OK**. Don't close the **Configure SQL Database** dialog yet.

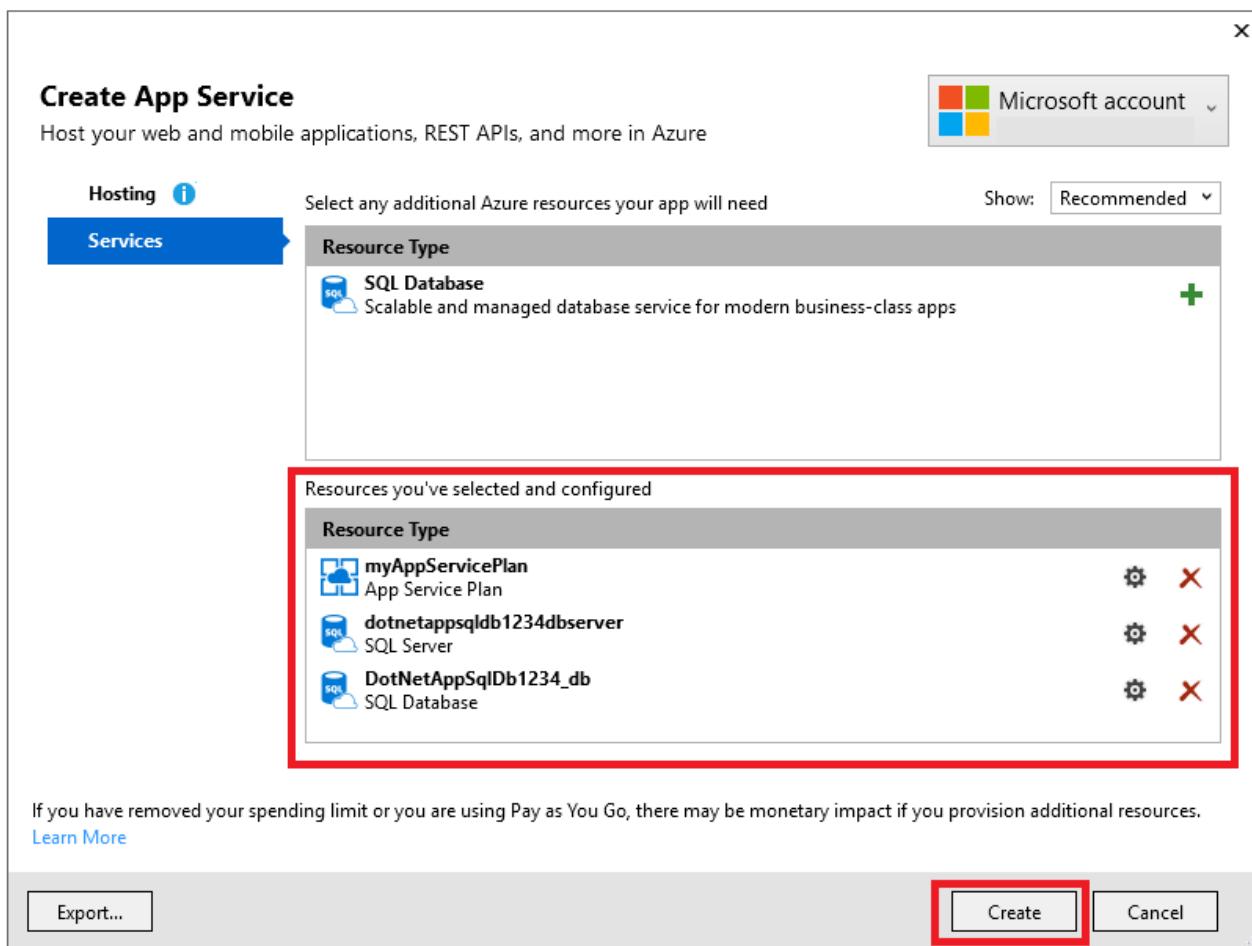
Create a SQL Database

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 created. 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**.

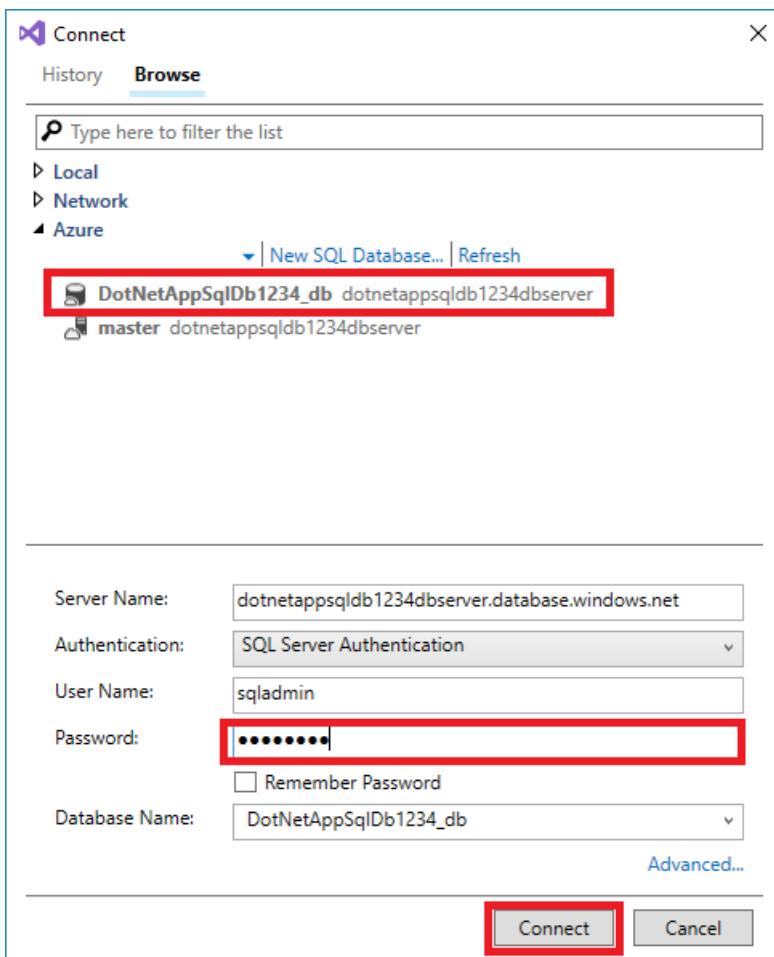
At the top of **SQL Server Object Explorer**, click the **Add SQL Server** button.

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**.

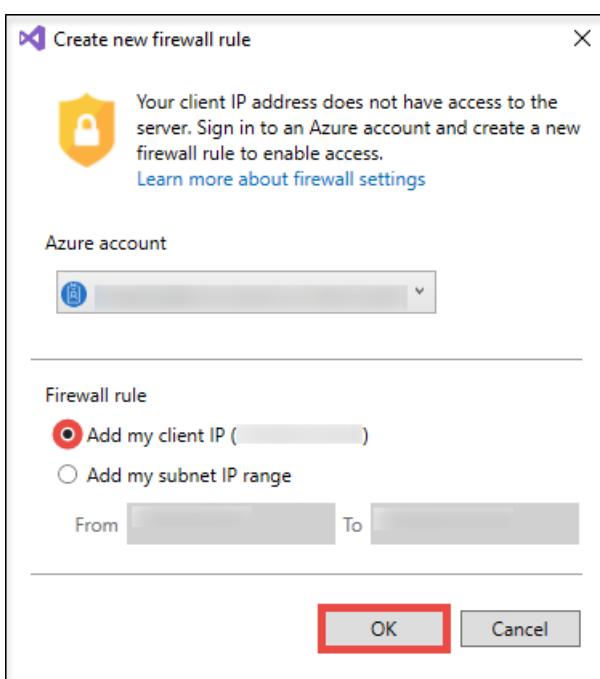


Allow client connection from your computer

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 web 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.

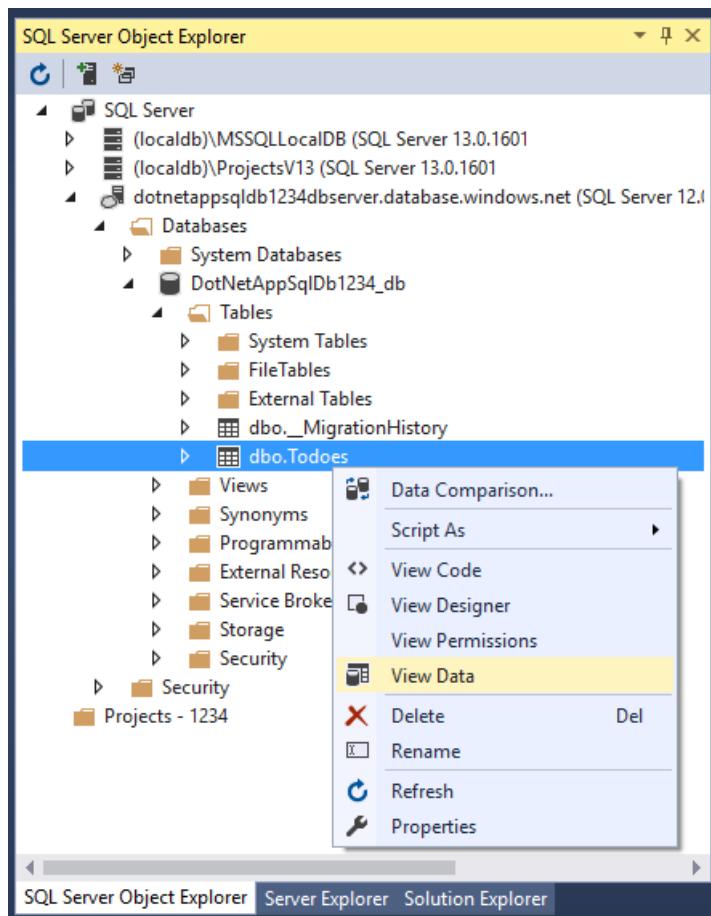
Make sure that **Add my client IP** is selected and click **OK**.



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 web 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.

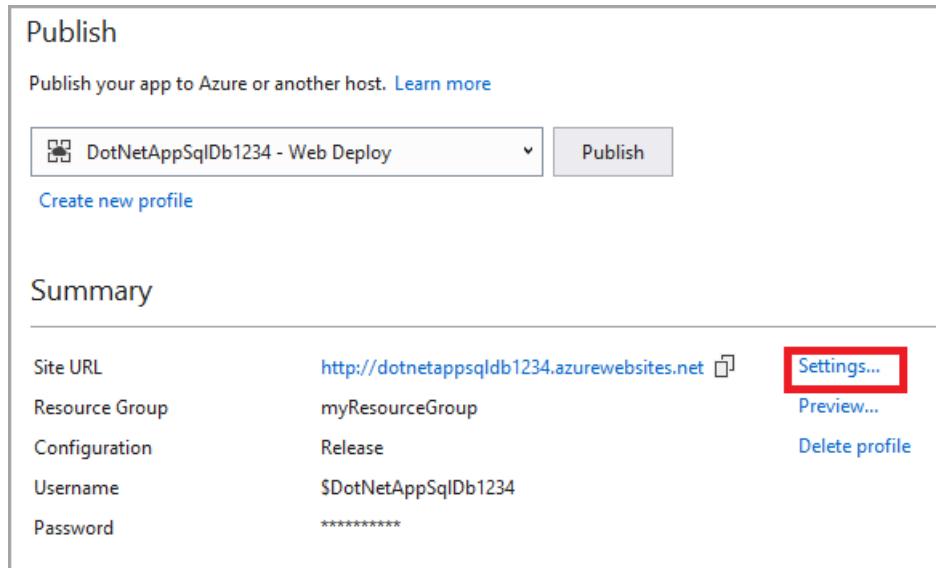
Remember that the `Edit` view doesn't show the `Done` field, because you didn't change the `Edit` view.

Enable Code First Migrations in Azure

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

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

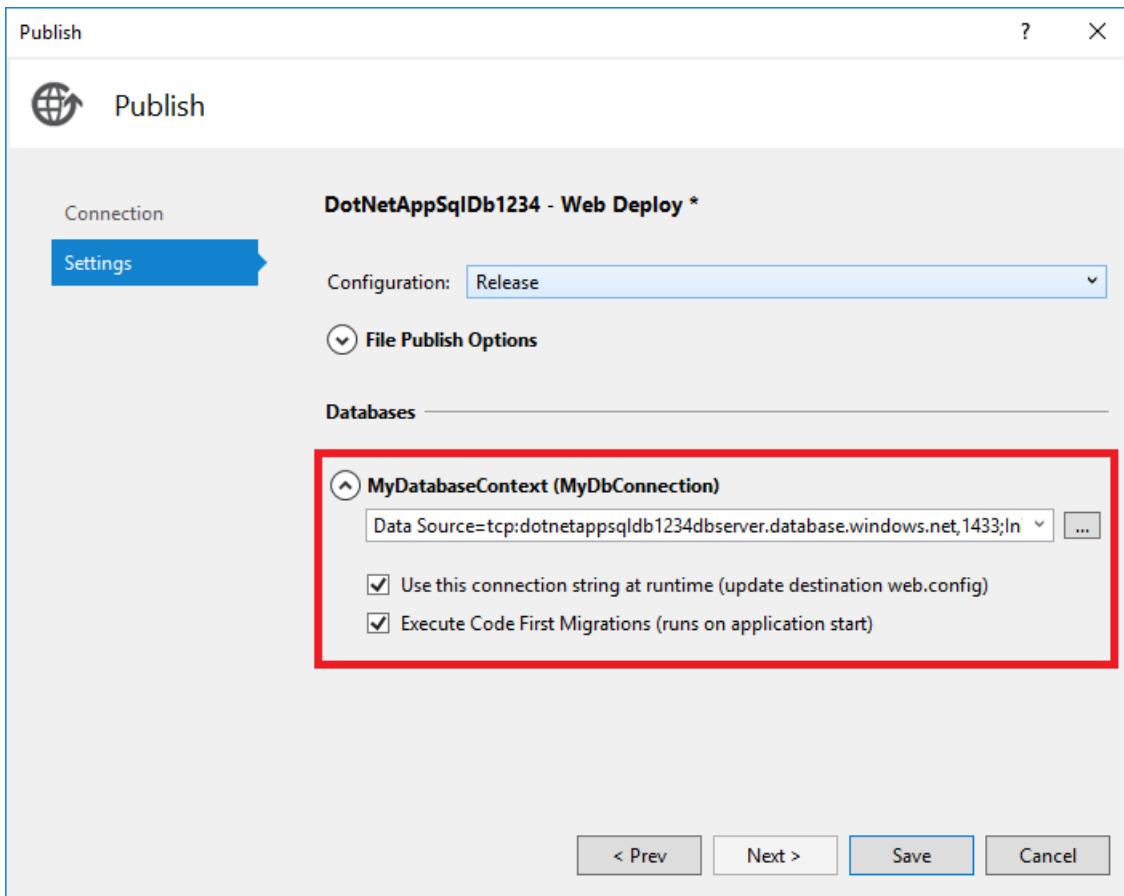
Click **Settings** to open the publish wizard.



In the wizard, click **Next**.

Make sure that the connection string for your SQL Database is populated in **MyDbContext** (**MyDbConnection**). 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 web 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	Action
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

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 web 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 web app.

Open Server Explorer

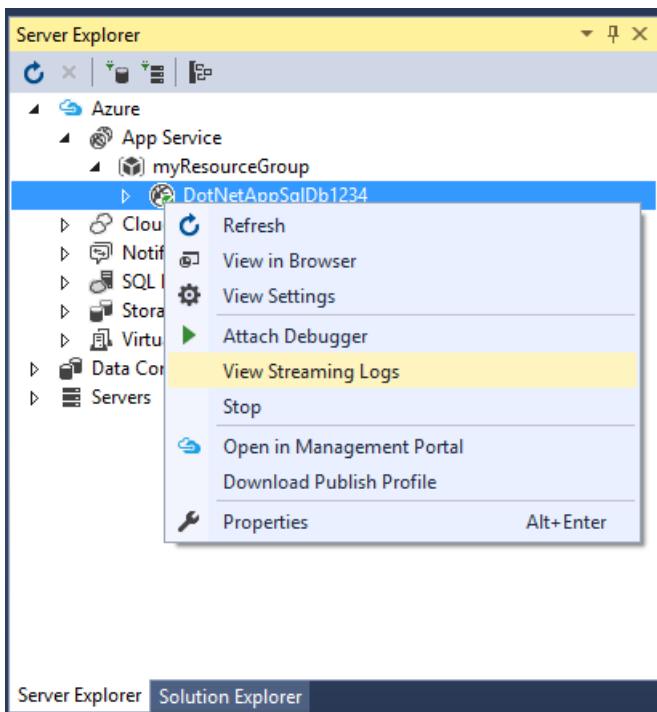
From the **View** menu, select **Server Explorer**. You can configure logging for your Azure web app in **Server Explorer**.

Enable log streaming

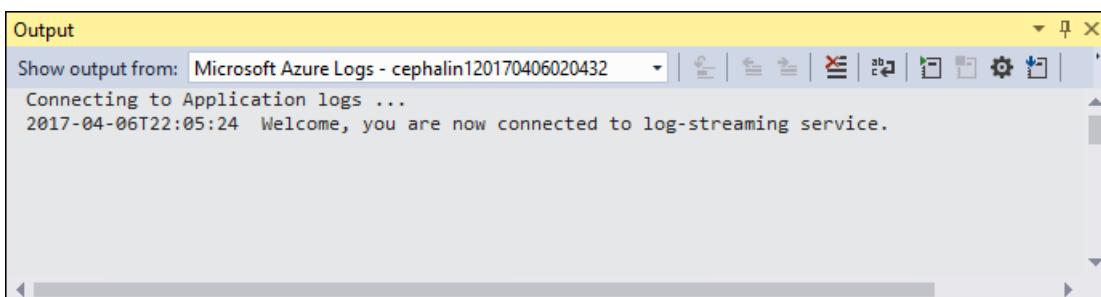
In **Server Explorer**, expand **Azure > App Service**.

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

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



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



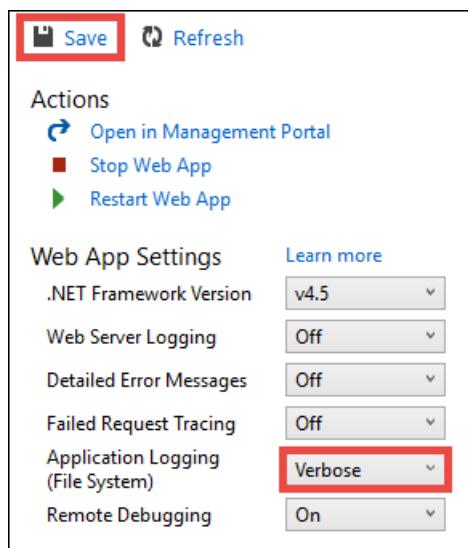
However, you don't see any of the trace messages yet. That's because when you first select **View Streaming Logs**, your Azure web app sets the trace level to `Error`, which only logs error events (with the `Trace.TraceError()` method).

Change trace levels

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

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

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



TIP

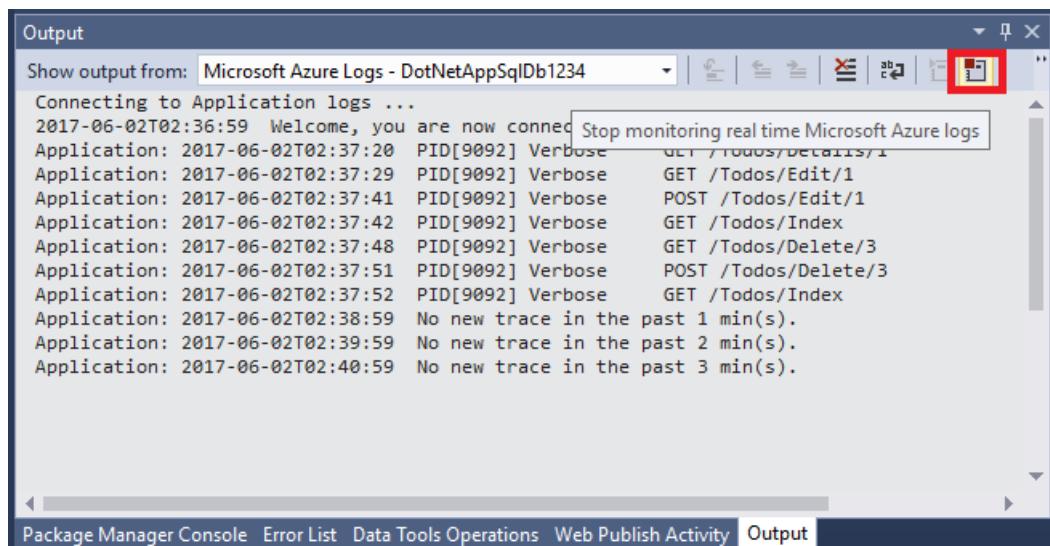
You can experiment with different trace levels to see what types of messages are displayed for each level. For example, the **Information** level includes all logs created by `Trace.TraceInformation()`, `Trace.TraceWarning()`, and `Trace.TraceError()`, but not logs created by `Trace.WriteLine()`.

In your browser navigate to your web 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 web app

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

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

The screenshot shows the 'App Services' blade in the Azure portal. The title bar says 'App Services' and 'rickaOrgName'. On the left, there's a vertical toolbar with icons for Add, Columns, Refresh, and other management tasks. The main area displays a table titled 'Subscriptions: All 2 selected'. The table has columns: NAME, RESOURCE GR..., STATUS, APP TYPE, and APP SERVICE PLAN. There is one item listed: 'DotNetAppSqlDb1234' (highlighted with a red box), which is part of 'myResourceGroup', 'Running', a 'Web app' under 'myAppServicePlan'.

You have landed in your web 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 'Overview' page for the web app 'DotNetAppSqlDb1234'. The left sidebar lists navigation options: Overview (highlighted in blue), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview)), Settings (Application settings, Authentication / Authorization, Backups, Custom domains). The main content area shows 'Essentials' details: Resource group (myResourceGroup), URL (http://dotnetappsqldb1234.azurewebsites.net), Status (Running), Location (West Europe), Subscription name (myAppServicePlan (Free)), MSDN, Subscription ID (74812a68-74812a68-74812a68-8422aec1). Below this is a 'Monitoring' section with a 'Requests and errors' chart. The chart shows two spikes in requests around 4:40 PM. The legend indicates blue for REQUESTS and red for HTTP SERVER ERRORS. The x-axis shows times from 4:15 PM to 5 PM.

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 map a custom DNS name to the web app.

[Map an existing custom DNS name to Azure Web Apps](#)

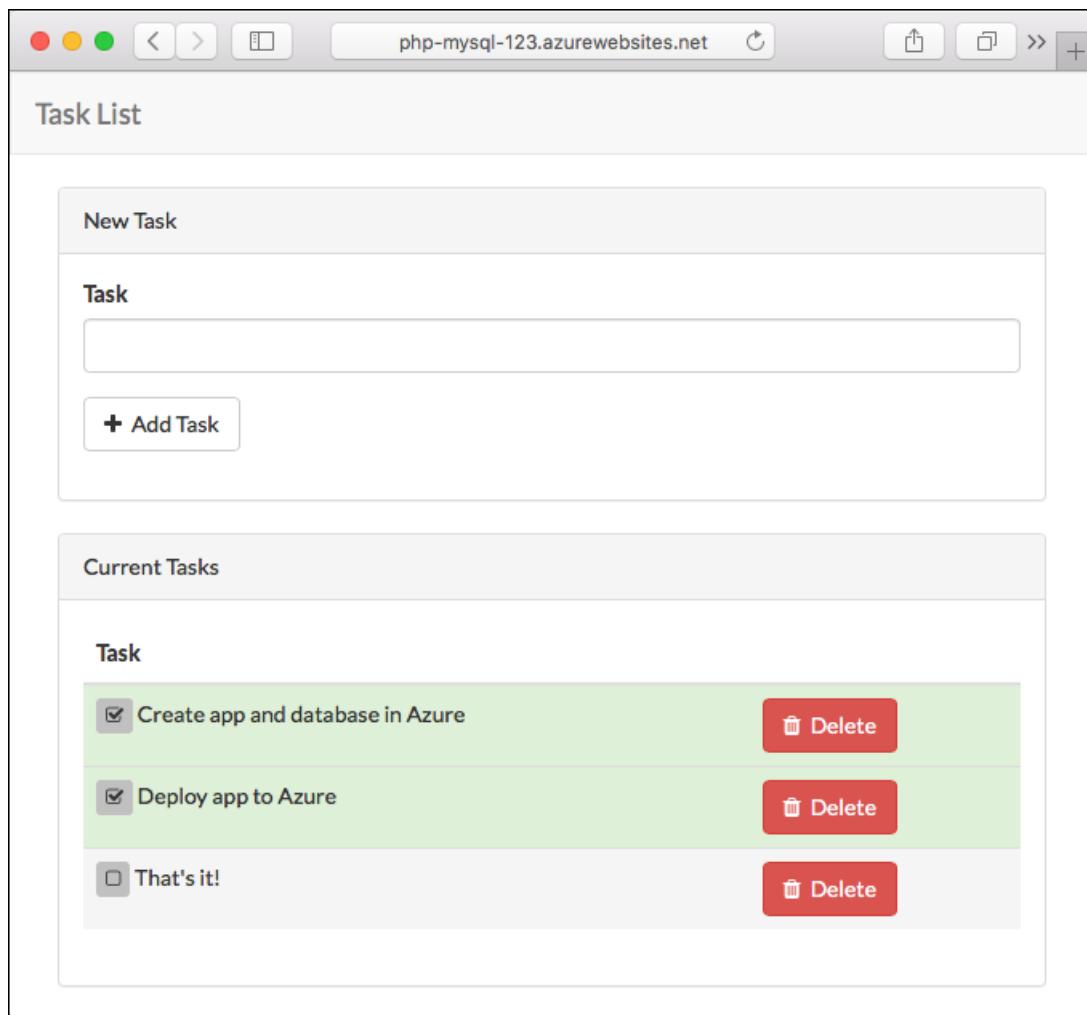
Build a PHP and MySQL web app in Azure

1/4/2018 • 17 min 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 web app in Azure App Service on Linux](#).

[Azure Web Apps](#) provides a highly scalable, self-patching web hosting service. This tutorial shows how to create a PHP web 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 Web Apps.



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

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](#)

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

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=sampled
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 "localhost". The interface is divided into two main sections: "New Task" and "Current Tasks".

New Task: Contains a text input field and a button labeled "+ Add Task".

Current Tasks: Contains three 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.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	

Create MySQL in Azure

In this step, you create a MySQL database in [Azure Database for MySQL \(Preview\)](#). Later, you configure the PHP application to connect to this database.

Create a resource group

In the Cloud Shell, create a resource group with the `az group create` command.

A [resource group](#) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service, run the `az appservice list-locations` command.

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

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

Create a MySQL server

In the Cloud Shell, create a server in Azure Database for MySQL (Preview) with the [az mysql server create](#) command.

In the following command, substitute your MySQL server name where you see the *<mysql_server_name>* placeholder (valid characters are `a-z`, `0-9`, and `-`). This name is part of the MySQL server's hostname (`<mysql_server_name>.database.windows.net`), it needs to be globally unique.

```
az mysql server create --name <mysql_server_name> --resource-group myResourceGroup --location "North Europe" --admin-user adminuser --admin-password My5up3r$tr0ngPa$w0rd!
```

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:

```
{
  "administratorLogin": "adminuser",
  "administratorLoginPassword": null,
  "fullyQualifiedDomainName": "<mysql_server_name>.database.windows.net",
  "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/<mysql_server_name>",
  "location": "northeurope",
  "name": "<mysql_server_name>",
  "resourceGroup": "myResourceGroup",
  ...
}
```

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.

```
az mysql server firewall-rule create --name allIPs --server <mysql_server_name> --resource-group
myResourceGroup --start-ip-address 0.0.0.0 --end-ip-address 255.255.255.255
```

NOTE

Azure Database for MySQL (Preview) doesn't currently limit connections only to Azure services. As IP addresses in Azure are dynamically assigned, it is better to enable all IP addresses. The service is in preview. Better methods for securing your database are planned.

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 *My5up3r\$tr0ngPa\$w0rd!*, which you specified when you created the database in Azure.

```
mysql -u adminuser@<mysql_server_name> -h <mysql_server_name>.database.windows.net -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 `samp1edb` database. Again, for simplicity of the tutorial, use *MySQLAzure2017* as the password.

```
CREATE USER 'phpappuser' IDENTIFIED BY 'MySQLAzure2017';
GRANT ALL PRIVILEGES ON samp1edb.* 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 (Preview).

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>.database.windows.net
DB_DATABASE=samp1edb
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 (Preview). 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 a `.pem` SSL certificate.

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/certificate.pem',
    ] : []
],
```

To learn how to generate this `certificate.pem`, see [Configure SSL connectivity in your application to securely connect to Azure Database for MySQL](#).

TIP

The path `/ssl/certificate.pem` points to an existing `certificate.pem` file in the Git repository. This file is provided for convenience in this tutorial. For best practice, you should not commit your `.pem` certificates into source control.

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 (Preview). 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.

New Task

Task

+ Add Task

Create app and database in Azure

Delete

Deploy data-driven app

Delete

That's it!

Delete

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

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

In the Cloud Shell, create deployment credentials with the [az webapp deployment user set](#) command. A deployment user is required for FTP and local Git deployment to a web app. The user name and password are account level. *They are different from your Azure subscription credentials.*

In the following example, replace `<username>` and `<password>` (including brackets) with a new user name and password. The user name must be unique. The password must be at least eight characters long, with two of the following three elements: letters, numbers, symbols.

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

If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error,

use a stronger password.

You create this deployment user only once; you can use it for all your Azure deployments.

NOTE

Record the user name and password. You need them to deploy the web app later.

Create an App Service plan

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

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)

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

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](#).

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --runtime  
"PHP|7.0" --deployment-local-git
```

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

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

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

NOTE

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

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

Configure database settings

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>.database.windows.net" 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 web 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 web 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 web 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

In the local terminal window, add an Azure remote to your local Git repository. Replace `<paste_copied_url_here>` with the URL of the Git remote that you saved from [Create a web app](#).

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

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

```
git push azure master
```

During deployment, Azure App Service communicates its progress with Git.

```
Counting objects: 3, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 291 bytes | 0 bytes/s, done.  
Total 3 (delta 2), reused 0 (delta 0)  
remote: Updating branch 'master'.  
remote: Updating submodules.  
remote: Preparing deployment for commit id 'a5e076db9c'.  
remote: Running custom deployment command...  
remote: Running deployment command...  
...  
< Output has been truncated for readability >
```

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 web 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 itself is a simple task management interface. It features a header 'Task List' and two main sections: 'New Task' and 'Current Tasks'. The 'New Task' section contains a text input field labeled 'Task' and a button labeled '+ Add Task'. The 'Current Tasks' section lists three items: 'Create app and database in Azure', 'Deploy app to Azure', and 'That's it!', each preceded by a small red 'Delete' button.

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.

New Task

Task

+ Add Task

Current Tasks

Task

<input checked="" type="checkbox"/> Create app and database in Azure	Delete
<input checked="" type="checkbox"/> Deploy data-driven app	Delete
<input type="checkbox"/> That's it!	Delete

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 web app and test the new functionality.

The screenshot shows a web application interface for managing tasks. At the top, the browser bar displays the URL "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 checkbox and 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 web 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 web app

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

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

The screenshot shows the Azure portal's 'App Services' blade. On the left is a vertical navigation bar with icons for App Services, Functions, Logic Apps, Container Registry, and App Configuration. The 'App Services' icon is highlighted with a red box. The main area is titled 'App Services' and shows a list of 'Subscriptions: All 2 selected'. A search bar 'Filter by name...' and a dropdown 'All subscriptions' are at the top. Below is a table with one item: 'NAME' (php-mysql-123), 'STATUS' (Running), 'APP TYPE' (Web app), 'APP SERVICE PLAN' (myAppServicePlan), 'LOCATION' (West Europe), and 'SUBSCRI...' (Visual Studio ...). The 'php-mysql-123' row is also highlighted with a red box.

You see your web 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 'Overview' page for the 'php-mysql-123' app service. The left sidebar has tabs for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, Deployment credentials, Deployment slots, and Deployment options. The 'Overview' tab is selected and highlighted with a blue box. The main content area has sections for 'Essentials' and 'Monitoring'. In the 'Essentials' section, it shows the 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. In the 'Monitoring' section, there's a 'Requests and errors' chart showing 4 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 a web app.

[Map an existing custom DNS name to Azure Web Apps](#)

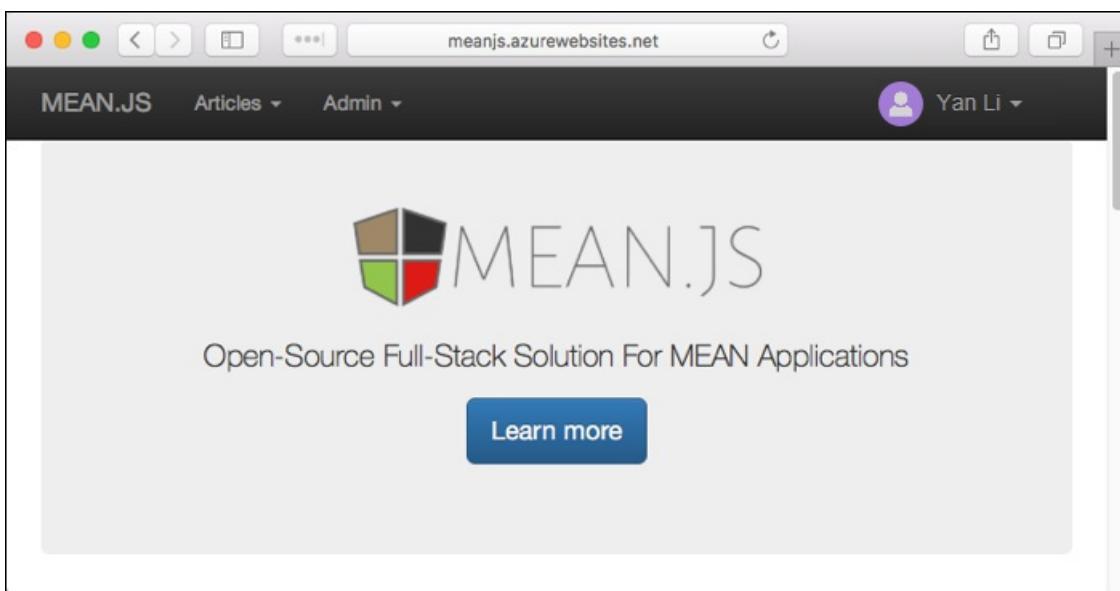
Build a Node.js and MongoDB web app in Azure

12/14/2017 • 14 min 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 web app in Azure App Service on Linux](#).

Azure Web Apps provides a highly scalable, self-patching web hosting service. This tutorial shows how to create a Node.js web app in Azure 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

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](#)

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

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
```

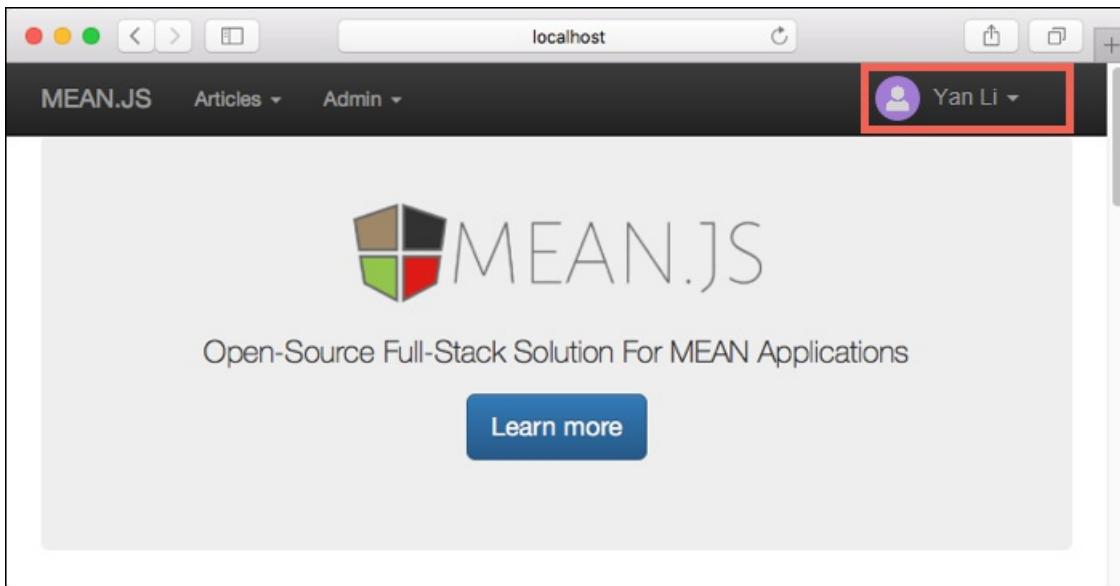
When the app is fully loaded, you see something similar to the following message:

```
-- 
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.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	A screenshot of the Azure Cloud Shell interface. On the left, there is a code block with the text "Click Try It in the upper right corner of a code block.". On the right, there is a toolbar with several icons: 'Azure CLI', 'Copy' (highlighted with a red box), and 'Try It' (also highlighted with a red box).
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	A screenshot of the Azure portal's top navigation bar. The 'Cloud Shell' button is highlighted with a red box among other icons like search, notifications, and help.

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

In the Cloud Shell, create a resource group with the `az group create` command.

A [resource group](#) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group named `myResourceGroup` in the *West Europe* location. To see all supported locations for App Service, run the `az appservice list-locations` command.

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

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

Create a Cosmos DB account

In the Cloud Shell, create a Cosmos DB account with the [az cosmosdb create](#) command.

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

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

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

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

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

Connect app to production 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":  
  "HvitsjIYz8TwRmIuPEUAAALRwqgK0zJUjW22wPL2U8zoMvhGvregBkBk9LdMTxqBgDETSq7obbwZtdeFY7hElTg==",  
  "secondaryMasterKey":  
  "Lu9aeZTiXU4PjuuyGBbvS1N9IRG3oegIrIh95U6V0stf9bJiiIpw3IfwSUgQWSEYM3VeEyrhHJ4rn3Ci0vuFqA==",  
  "secondaryReadonlyMasterKey":  
  "LpsCicpVZqHy7qbMgrzbRKjbYCwCKPQRl0QpgReAOxMcggTvxJFA94fTi0oQ7xtxpftTJcXkjTirQ0pT7QFrQ=="  
}
```

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

In the Cloud Shell, create deployment credentials with the `az webapp deployment user set` command. A

deployment user is required for FTP and local Git deployment to a web app. The user name and password are account level. *They are different from your Azure subscription credentials.*

In the following example, replace <username> and <password> (including brackets) with a new user name and password. The user name must be unique. The password must be at least eight characters long, with two of the following three elements: letters, numbers, symbols.

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

If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

You create this deployment user only once; you can use it for all your Azure deployments.

NOTE

Record the user name and password. You need them to deploy the web app later.

Create an App Service plan

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

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)

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

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 `NODE|6.9`. To see all supported runtimes, run [az webapp list-runtimes](#).

```
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'll 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 web 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 web 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

In the local terminal window, add an Azure remote to your local Git repository. This Azure remote was created for you in [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 prompted for a password, make sure that you enter the password you created in [Configure a deployment user](#), not the password you use to log 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 web app at any point, App Service doesn't rerun these automation tasks.

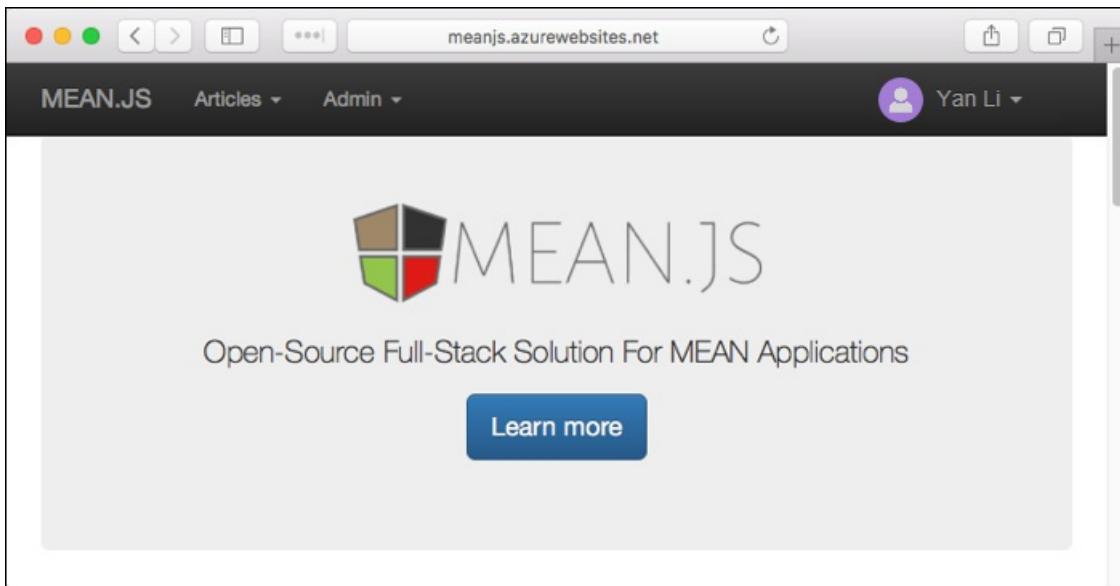
Browse to the Azure web app

Browse to the deployed web 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:

```
var 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) {
  var 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.

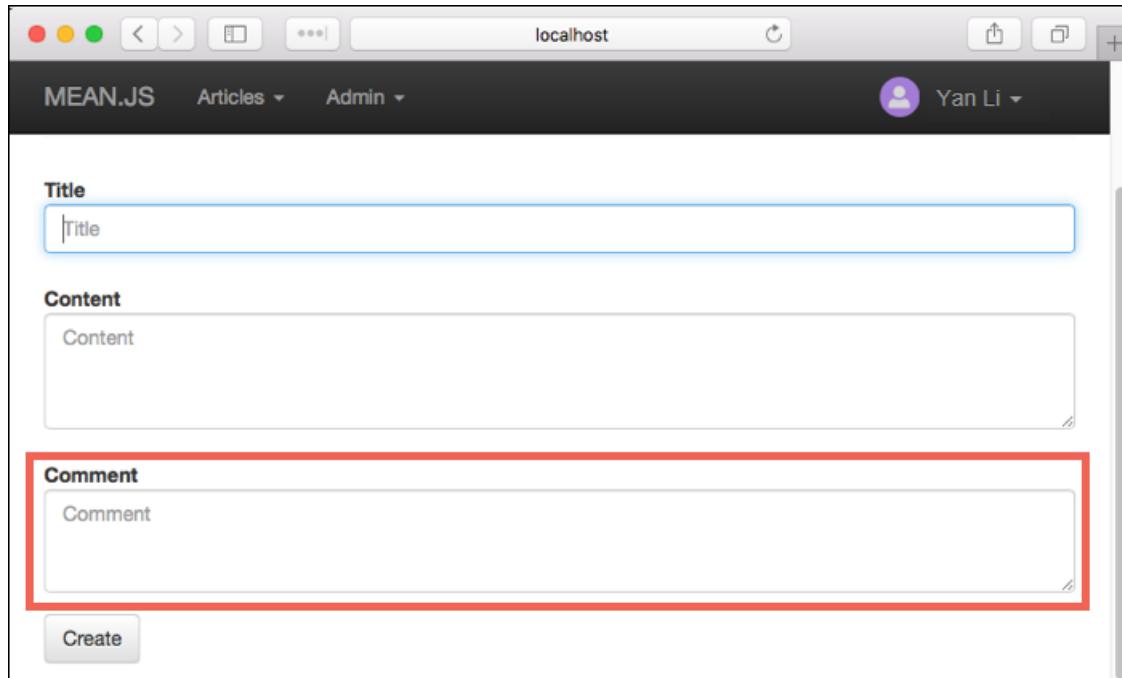
```
# Bash
gulp prod
NODE_ENV=production node server.js

# 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 web app and try out the new functionality.

The screenshot shows a web application interface for creating a new article. The top navigation bar includes links for 'MEAN.JS', 'Articles', and 'Admin', along with a user profile for 'Yan Li'. The main form has three fields: 'Title' (a text input field), 'Content' (a large text area), and 'Comment' (another text area, which is highlighted with a red border). A 'Create' button is located at the bottom of the form.

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 web 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 web app

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

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

The screenshot shows the Azure App Services portal. On the left, there's a sidebar with icons for App Services (selected), Storage, Functions, Logic Apps, Container Registry, and App Configuration. The main area has a title bar 'App Services Microsoft'. Below it, there are buttons for '+ Add', 'Columns', and 'Refresh'. A message says 'Subscriptions: All 28 selected – Don't see a subscription? Switch directories'. There's a search bar 'Filter by name...' and a link 'All subscriptions'. A table lists one item: 'meanjs' (Status: Running, App Type: Web app, App Service Plan: myAppServicePlan, Location: West Europe). The 'meanjs' row is highlighted with a red box.

By default, the portal shows your web 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 'Overview' (selected), 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'DEPLOYMENT' (with 'Quickstart' and 'Deployment credentials'), and 'More'. The main content area has a toolbar with 'Browse', 'Stop', 'Swap', 'Restart', 'Delete', and 'More'. It shows the 'Essentials' section with details: Resource group (myResourceGroup), Status (Running), Location (West Europe), Subscription name (myAppServicePlan), URL (http://meanjs.azurewebsites.net), App Service plan/pricing tier (myAppServicePlan (Free)), Git/Deployment username (redacted), Git clone url (redacted), Subscription ID (redacted), and FTP hostname (ftp://waws-prod-am2-025.ftp.azurewebsites.net). Below that is the 'Monitoring' section with '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 your web app.

[Map an existing custom DNS name to Azure Web Apps](#)

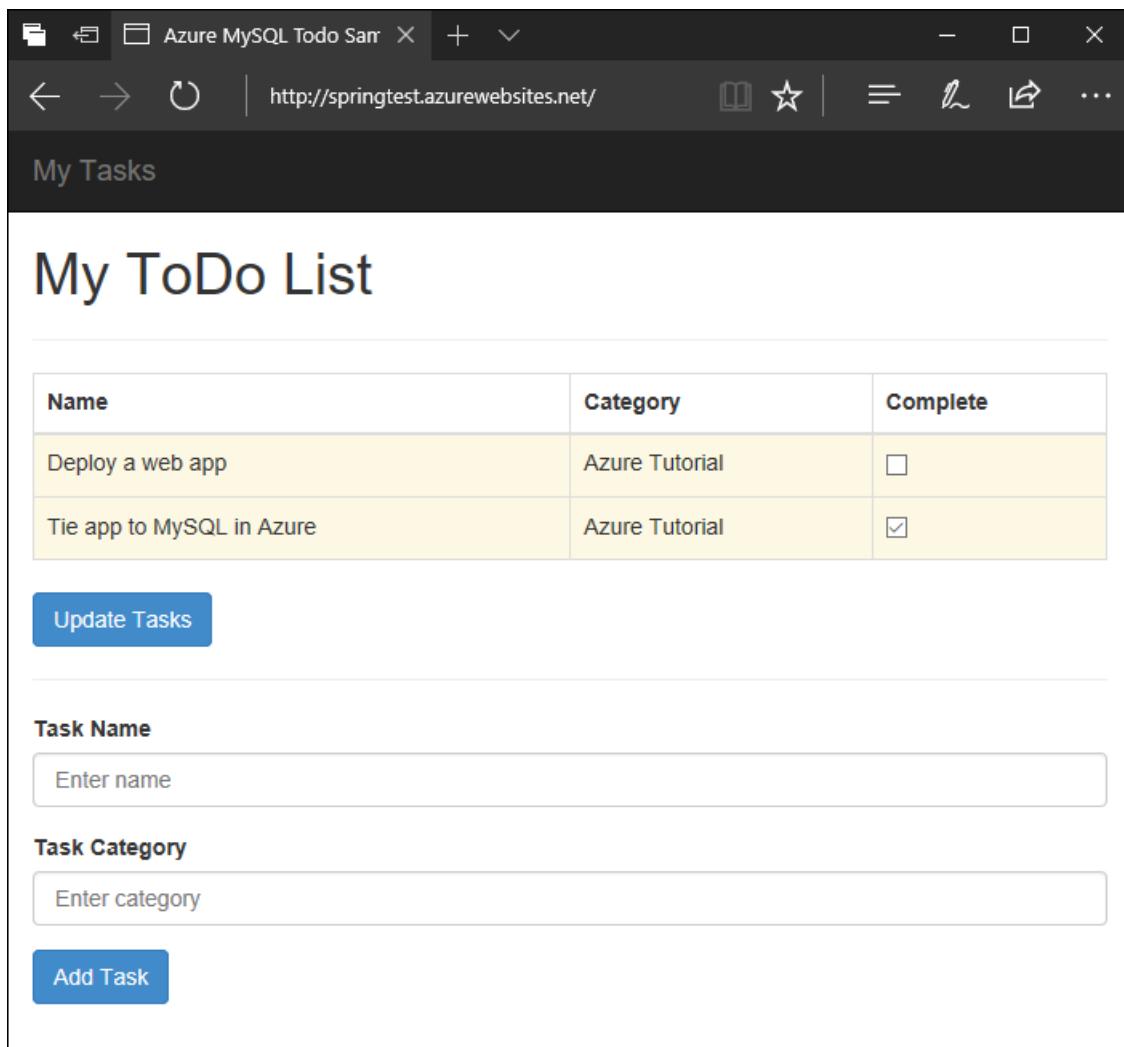
Build a Java and MySQL web app in Azure

12/14/2017 • 10 min to read • [Edit Online](#)

NOTE

This article deploys an app to App Service on Windows. To deploy to App Service on *Linux*, see [Deploy a containerized Spring Boot app to Azure](#).

This tutorial shows you how to create a Java web app in Azure and connect it to a MySQL database. When you are finished, you will have a [Spring Boot](#) application storing data in [Azure Database for MySQL](#) running on [Azure App Service Web Apps](#).



The screenshot shows a browser window titled "Azure MySQL Todo Sam". The address bar displays the URL <http://springtest.azurewebsites.net/>. The page content is titled "My Tasks" and features a heading "My ToDo List". Below the heading is a table listing two tasks:

Name	Category	Complete
Deploy a web app	Azure Tutorial	<input type="checkbox"/>
Tie app to MySQL in Azure	Azure Tutorial	<input checked="" type="checkbox"/>

Below the table is a blue button labeled "Update Tasks". Further down the page are input fields for "Task Name" (with placeholder "Enter name") and "Task Category" (with placeholder "Enter category"), followed by a blue "Add Task" button.

In this tutorial, you learn how to:

- Create a MySQL database in Azure
- Connect a sample app to the database
- Deploy the app to Azure
- Update and redeploy the app
- Stream diagnostic logs from Azure
- Monitor the app in the Azure portal

Prerequisites

1. [Download and install Git](#)
2. [Download and install the Java 7 JDK or above](#)
3. [Download, install, and start MySQL](#)

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

Prepare local MySQL

In this step, you create a database in a local MySQL server for use in testing the app locally on your machine.

Connect to 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 already running. If not, make sure that your local MySQL server is started by following the [MySQL post-installation steps](#).

Create a database

In the `mysql` prompt, create a database and a table for the to-do items.

```
CREATE DATABASE tododb;
```

Exit your server connection by typing `quit`.

```
quit
```

Create and run the sample app

In this step, you clone sample Spring boot app, configure it to use the local MySQL database, and run it on your computer.

Clone the sample

In the terminal window, navigate to a working directory and clone the sample repository.

```
git clone https://github.com/azure-samples/mysql-spring-boot-todo
```

Configure the app to use the MySQL database

Update the `spring.datasource.password` and value in `spring-boot-mysql-todo/src/main/resources/application.properties` with the same root password used to open the MySQL prompt:

```
spring.datasource.password=mysqlpass
```

Build and run the sample

Build and run the sample using the Maven wrapper included in the repo:

```
cd spring-boot-mysql-todo  
mvnw package spring-boot:run
```

Open your browser to <http://localhost:8080> to see in the sample in action. As you add tasks to the list, use the following SQL commands in the MySQL prompt to view the data stored in MySQL.

```
use testdb;  
select * from todo_item;
```

Stop the application by hitting `Ctrl + C` in the terminal.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	

Create an Azure MySQL database

In this step, you create an [Azure Database for MySQL](#) instance using the [Azure CLI](#). You configure the sample application to use this database later on in the tutorial.

Create a resource group

Create a [resource group](#) with the `az group create` command. An Azure resource group is a logical container where related resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group in the North Europe region:

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

To see the possible values you can use for `--location`, use the `az appservice list-locations` command.

Create a MySQL server

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

Substitute your own unique MySQL server name where you see the `<mysql_server_name>` placeholder. This name is part of your MySQL server's hostname, `<mysql_server_name>.mysql.database.azure.com`, so it needs to be globally unique. Also substitute `<admin_user>` and `<admin_password>` with your own values.

```
az mysql server create --name <mysql_server_name> --resource-group myResourceGroup --location "North Europe" --admin-user <admin_user> --admin-password <admin_password>
```

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

```
{  
    "administratorLogin": "admin_user",  
    "administratorLoginPassword": null,  
    "fullyQualifiedDomainName": "mysql_server_name.mysql.database.azure.com",  
    "id": "/subscriptions/00000000-0000-0000-0000-  
0000000000/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/mysql_server_name",  
    "location": "northeurope",  
    "name": "mysql_server_name",  
    "resourceGroup": "mysqlJavaResourceGroup",  
    ...  
    < 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.

```
az mysql server firewall-rule create --name allIPs --server <mysql_server_name> --resource-group  
myResourceGroup --start-ip-address 0.0.0.0 --end-ip-address 255.255.255.255
```

NOTE

Azure Database for MySQL (Preview) does not currently automatically enable connections from Azure services. As IP addresses in Azure are dynamically assigned, it is better to enable all IP addresses for now. As the service continues its preview, better methods for securing your database will be enabled.

Configure the Azure MySQL database

In the local terminal window, connect to the MySQL server in Azure. Use the value you specified previously for `<admin_user>` and `<mysql_server_name>`.

```
mysql -u <admin_user>@<mysql_server_name> -h <mysql_server_name>.mysql.database.azure.com -P 3306 -p
```

Create a database

In the `mysql` prompt, create a database and a table for the to-do items.

```
CREATE DATABASE tododb;
```

Create a user with permissions

Create a database user and give it all privileges in the `tododb` database. Replace the placeholders `<Javaapp_user>` and `<Javaapp_password>` with your own unique app name.

```
CREATE USER '<Javaapp_user>' IDENTIFIED BY '<Javaapp_password>';  
GRANT ALL PRIVILEGES ON tododb.* TO '<Javaapp_user>';
```

Exit your server connection by typing `quit`.

```
quit
```

Deploy the sample to Azure App Service

Create an Azure App Service plan with the **FREE** pricing tier using the [az appservice plan create](#) CLI command. The appservice plan defines the physical resources used to host your apps. All applications assigned to an appservice plan share these resources, allowing you to save cost when hosting multiple apps.

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

When the plan is ready, the Azure CLI shows similar output to the following example:

```
{  
    "adminSiteName": null,  
    "appServicePlanName": "myAppServicePlan",  
    "geoRegion": "North Europe",  
    "hostingEnvironmentProfile": null,  
    "id": "/subscriptions/0000-  
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",  
    "kind": "app",  
    "location": "North Europe",  
    "maximumNumberOfWorkers": 1,  
    "name": "myAppServicePlan",  
    ...  
    < Output has been truncated for readability >  
}
```

Create an Azure Web app

In the Cloud Shell, use the [az webapp create](#) CLI command to create a web app definition in the `myAppServicePlan` App Service plan. The web app definition provides a URL to access your application with and configures several options to deploy your code to Azure.

```
az webapp create --name <app_name> --resource-group myResourceGroup --plan myAppServicePlan
```

Substitute the `<app_name>` placeholder with your own unique app name. This unique name is part of the default domain name for the web app, so the name needs to be unique across all apps in Azure. You can map a custom domain name entry to the web app before you expose it to your users.

When the web app definition is ready, the Azure CLI shows information similar to the following example:

```
{  
    "availabilityState": "Normal",  
    "clientAffinityEnabled": true,  
    "clientCertEnabled": false,  
    "cloningInfo": null,  
    "containerSize": 0,  
    "dailyMemoryTimeQuota": 0,  
    "defaultHostName": "<app_name>.azurewebsites.net",  
    "enabled": true,  
    ...  
    < Output has been truncated for readability >  
}
```

Configure Java

In the Cloud Shell, set up the Java runtime configuration that your app needs with the [az appservice web config update](#) command.

The following command configures the web app to run on a recent Java 8 JDK and [Apache Tomcat 8.0](#).

```
az webapp config set --name <app_name> --resource-group myResourceGroup --java-version 1.8 --java-container Tomcat --java-container-version 8.0
```

Configure the app to use the Azure SQL database

Before running the sample app, set application settings on the web app to use the Azure MySQL database you created in Azure. These properties are exposed to the web application as environment variables and override the values set in the application.properties inside the packaged web app.

In the Cloud Shell, set application settings using `az webapp config appsettings` in the CLI:

```
az webapp config appsettings set --settings SPRING_DATASOURCE_URL="jdbc:mysql://<mysql_server_name>.mysql.database.azure.com:3306/tododb?verifyServerCertificate=true&useSSL=true&requireSSL=false" --resource-group myResourceGroup --name <app_name>
```

```
az webapp config appsettings set --settings SPRING_DATASOURCE_USERNAME=Javaapp_user@mysql_server_name --resource-group myResourceGroup --name <app_name>
```

```
az webapp config appsettings set --settings SPRING_DATASOURCE_PASSWORD=Javaapp_password --resource-group myResourceGroup --name <app_name>
```

Get FTP deployment credentials

You can deploy your application to Azure appservice in various ways including FTP, local Git, GitHub, Visual Studio Team Services, and BitBucket. For this example, FTP to deploy the .WAR file built previously on your local machine to Azure App Service.

To determine what credentials to pass along in an `ftp` command to the Web App, Use [az appservice web deployment list-publishing-profiles](#) command in the Cloud Shell:

```
az webapp deployment list-publishing-profiles --name <app_name> --resource-group myResourceGroup --query "[?publishMethod=='FTP'].{URL:publishUrl, Username:userName, Password:userPWD}" --output json
```

```
[  
 {  
 "Password": "aBcDeFgHiJkLmNoPqRsTuVwXyZ",  
 "URL": "ftp://waws-prod-blu-069.ftp.azurewebsites.windows.net/site/wwwroot",  
 "Username": "app_name\\$app_name"  
 }  
]
```

Upload the app using FTP

Use your favorite FTP tool to deploy the .WAR file to the `/site/wwwroot/webapps` folder on the server address taken from the `URL` field in the previous command. Remove the existing default (ROOT) application directory and replace the existing ROOT.war with the .WAR file built in the earlier in the tutorial.

```
ftp waws-prod-blu-069.ftp.azurewebsites.windows.net
Connected to waws-prod-blu-069.drip.azurewebsites.windows.net.
220 Microsoft FTP Service
Name (waws-prod-blu-069.ftp.azurewebsites.windows.net:raisa): app_name\${app_name}
331 Password required
Password:
cd /site/wwwroot/webapps
mdelete -i ROOT/*
rmdir ROOT/
put target/TodoDemo-0.0.1-SNAPSHOT.war ROOT.war
```

Test the web app

Browse to http://<app_name>.azurewebsites.net/ and add a few tasks to the list.

The screenshot shows a browser window titled "Azure MySQL Todo Sam". The address bar contains the URL "http://springtest.azurewebsites.net/". The main content area is titled "My Tasks" and displays a heading "My ToDo List". Below the heading is a table with three rows:

Name	Category	Complete
Deploy a web app	Azure Tutorial	<input type="checkbox"/>
Tie app to MySQL in Azure	Azure Tutorial	<input checked="" type="checkbox"/>

Below the table is a blue "Update Tasks" button. Further down, there are input fields for "Task Name" (with placeholder "Enter name") and "Task Category" (with placeholder "Enter category"). At the bottom is a blue "Add Task" button.

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

Update the app and redeploy

Update the application to include an additional column in the todo list for what day the item was created. Spring Boot handles updating the database schema for you as the data model changes without altering your existing database records.

1. On your local system, open up `src/main/java/com/example/fabrikam/TodoItem.java` and add the following imports to the class:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
```

2. Add a `String` property `timeCreated` to `src/main/java/com/example/fabrikam/TodoItem.java`, initializing it with a timestamp at object creation. Add getters/setters for the new `timeCreated` property while you are editing this file.

```
private String name;
private boolean complete;
private String timeCreated;
...

public TodoItem(String category, String name) {
    this.category = category;
    this.name = name;
    this.complete = false;
    this.timeCreated = new SimpleDateFormat("MMMM dd, YYYY").format(Calendar.getInstance().getTime());
}
...
public void setTimeCreated(String timeCreated) {
    this.timeCreated = timeCreated;
}

public String getTimeCreated() {
    return timeCreated;
}
```

3. Update `src/main/java/com/example/fabrikam/TodoDemoController.java` with a line in the `updateTodo` method to set the timestamp:

```
item.setComplete(requestItem.isComplete());
item.setId(requestItem.getId());
item.setTimeCreated(requestItem.getTimeCreated());
repository.save(item);
```

4. Add support for the new field in the Thymeleaf template. Update `src/main/resources/templates/index.html` with a new table header for the timestamp, and a new field to display the value of the timestamp in each table data row.

```
<th>Name</th>
<th>Category</th>
<th>Time Created</th>
<th>Complete</th>
...
<td th:text="${item.category}">item_category</td><input type="hidden" th:field="*
{todoList[__${i.index}__].category}"/>
<td th:text="${item.timeCreated}">item_time_created</td><input type="hidden" th:field="*
{todoList[__${i.index}__].timeCreated}"/>
<td><input type="checkbox" th:checked="${item.complete} == true" th:field="*
{todoList[__${i.index}__].complete}"/></td>
```

5. Rebuild the application:

```
mvnw clean package
```

6. FTP the updated .WAR as before, removing the existing `site/wwwroot/webapps/ROOT` directory and `ROOT.war`, then uploading the updated .WAR file as `ROOT.war`.

When you refresh the app, a **Time Created** column is now visible. When you add a new task, the app will populate the timestamp automatically. Your existing tasks remain unchanged and work with the app even though the underlying data model has changed.

Name	Category	Time Created	Complete
Deploy a web app	Azure Tutorial		<input type="checkbox"/>
Tie app to MySQL in Azure	Azure Tutorial		<input checked="" type="checkbox"/>
Add date category to app	Azure Tutorial	June 01, 2017	<input type="checkbox"/>

Stream diagnostic logs

While your Java application runs in Azure App Service, you can get the console logs piped directly 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
```

Manage your Azure web app

Go to the Azure portal to see the web app you created.

To do this, sign in to <https://portal.azure.com>.

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

The screenshot shows the Azure App Services blade. At the top, there are buttons for 'Add', 'Columns', 'Refresh', 'Start', 'Restart', 'Stop', and 'Delete'. Below this, a section titled 'Subscriptions' shows 'Visual Studio Enterprise'. A search bar 'Filter by name...' and dropdowns for 'All locations' and 'No grouping' are present. A table lists '2 items': 'springDemoApp' (Status: Running, App Type: Web app, App Service Plan: myAppServicePlan, Location: East US, Subscription: Visual Studio Ent...). The left sidebar has icons for various services like Storage, Blob, Queue, and Table.

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

The screenshot shows the Overview blade for the 'springDemoApp' web app. The top navigation bar includes a search bar, 'Browse', 'Stop', 'Swap', 'Restart', 'Delete', and a 'More' button. A purple banner says 'Click here to access our Quickstart guide for deploying code to your app'. The main area is divided into sections: 'Essentials' (Resource group: myResourceGroup, Status: Running, Location: East US, Subscription name: Visual Studio Enterprise), 'Monitoring' (Requests and errors), and 'Deployment' (Quickstart, Deployment credentials, Deployment slots). The left sidebar lists 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', and 'DEPLOYMENT' (Quickstart, Deployment credentials, Deployment slots).

These tabs in the blade show the many great features you can add to your web app. The following list gives you just a few of the possibilities:

- Map a custom DNS name
- Bind a custom SSL certificate
- Configure continuous deployment
- Scale up and out
- Add user authentication

Clean up resources

If you don't need these resources for another tutorial (see [Next steps](#)), you can delete them by running the following command in the Cloud Shell:

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

Next steps

- Create a MySQL database in Azure
- Connect a sample Java app to the MySQL
- Deploy the app to Azure
- Update 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 Web Apps](#)

Build a Node.js RESTful API and deploy it to an API app in Azure

12/14/2017 • 8 min to read • [Edit Online](#)

This quickstart shows how to create a REST API, written with Node.js [Express](#), using a [Swagger](#) definition and deploying it on Azure. You create the app using command-line tools, configure resources with the [Azure CLI](#), and deploy the app using Git. When you've finished, you have a working sample REST API running on Azure.

Prerequisites

- [Git](#)
- [Node.js and NPM](#)

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

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click Try It in the upper right corner of a code block.	
Click the Cloud Shell button on the menu in the upper right of the Azure portal .	

If you choose to install and use the CLI locally, this topic requires that you are running the Azure CLI version 2.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI 2.0](#).

Prepare your environment

1. In a terminal window, run the following command to clone the sample to your local machine.

```
git clone https://github.com/Azure-Samples/app-service-api-node-contact-list
```

2. Change to the directory that contains the sample code.

```
cd app-service-api-node-contact-list
```

3. Install [Swaggerize](#) on your local machine. Swaggerize is a tool that generates Node.js code for your REST API from a Swagger definition.

```
npm install -g yo  
npm install -g generator-swaggerize
```

Generate Node.js code

This section of the tutorial models an API development workflow in which you create Swagger metadata first and use that to scaffold (auto-generate) server code for the API.

Change directory to the `start` folder, then run `yo swaggerize`. Swaggerize creates a Nodejs project for your API from the Swagger definition in `api.json`.

```
cd start
yo swaggerize --apiPath api.json --framework express
```

When Swaggerize asks for a project name, use `ContactList`.

```
Swaggerize Generator
Tell us a bit about your application
? What would you like to call this project: ContactList
? Your name: Francis Totten
? Your github user name: fabfrank
? Your email: frank@fabrikam.net
```

Customize the project code

1. Copy the `lib` folder into the `ContactList` folder created by `yo swaggerize`, then change directory into `ContactList`.

```
cp -r lib/ ContactList/
cd ContactList
```

2. Install the `jsonpath` and `swaggerize-ui` NPM modules.

```
npm install --save jsonpath swaggerize-ui
```

3. Replace the code in the `handlers/contacts.js` with the following code:

```
'use strict';

var repository = require('../lib/contactRepository');

module.exports = {
  get: function contacts_get(req, res) {
    res.json(repository.all())
  }
};
```

This code uses the JSON data stored in `lib/contacts.json` served by `lib/contactRepository.js`. The new `contacts.js` code returns all contacts in the repository as a JSON payload.

4. Replace the code in the `handlers/contacts/{id}.js` file with the following code:

```
'use strict';

var repository = require('../lib/contactRepository');

module.exports = {
    get: function contacts_get(req, res) {
        res.json(repository.get(req.params['id']));
    }
};
```

This code lets you use a path variable to return only the contact with a given ID.

- Replace the code in **server.js** with the following code:

```
'use strict';

var port = process.env.PORT || 8000;

var http = require('http');
var express = require('express');
var bodyParser = require('body-parser');
var swaggerize = require('swaggerize-express');
var swaggerUi = require('swaggerize-ui');
var path = require('path');
var fs = require("fs");

fs.existsSync = fs.existsSync || require('path').existsSync;

var app = express();

var server = http.createServer(app);

app.use(bodyParser.json());

app.use(swaggerize({
    api: path.resolve('./config/swagger.json'),
    handlers: path.resolve('./handlers'),
    docspath: '/swagger'
}));

// change four
app.use('/docs', swaggerUi({
    docs: '/swagger'
}));

server.listen(port, function () {
```

This code makes some small changes to let it work with Azure App Service and exposes an interactive web interface for your API.

Test the API locally

- Start up the Node.js app

```
npm start
```

- Browse to <http://localhost:8000/contacts> to view the JSON for the entire contact list.

```
{
  "id": 1,
  "name": "Barney Poland",
  "email": "barney@contoso.com"
},
{
  "id": 2,
  "name": "Lacy Barrera",
  "email": "lacy@contoso.com"
},
{
  "id": 3,
  "name": "Lora Riggs",
  "email": "lora@contoso.com"
}
```

3. Browse to <http://localhost:8000/contacts/2> to view the contact with an `id` of two.

```
{
  "id": 2,
  "name": "Lacy Barrera",
  "email": "lacy@contoso.com"
}
```

4. Test the API using the Swagger web interface at <http://localhost:8000/docs>.

The screenshot shows the Swagger UI interface for a 'Contact List' API. The main title is 'Contact List' with a subtitle 'A Contact list API based on Swagger and built using Node.js'. Under the 'Contacts' section, there are two operations: 'GET /contacts' and 'GET /contacts/{id}'. The 'GET /contacts/{id}' operation is currently selected, with the path parameter 'id' set to '2'. The 'Response Class (Status 200)' section shows a JSON schema for the response body:

```
[  
  null  
]
```

The 'Response Content Type' is set to 'application/json'. The 'Parameters' table shows the 'id' parameter with a value of '2', type 'path', and data type 'integer'. There is a 'Try it out!' button and a 'Curl' section with the command:

```
curl -X GET --header "Accept: application/json" "http://localhost:8000/contacts/2"
```

The 'Request URL' is listed as 'http://localhost:8000/contacts/2'. The 'Response Body' section shows the JSON response:

```
[  
  {  
    "id": 2,  
    "name": "Lacy Barrera",  
    "email": "lacy@contoso.com"  
  }]
```

Create an API App

In this section, you use the Azure CLI 2.0 to create the resources to host the API on Azure App Service.

1. Log in to your Azure subscription with the [az login](#) command and follow the on-screen directions.

```
az login
```

2. If you have multiple Azure subscriptions, change the default subscription to the desired one.

```
az account set --subscription <name or id>
```

3. Create a resource group with the [az group create](#) command.

A [resource group](#) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

The following example creates a resource group named *myResourceGroup* in the *westeurope* location.

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

To see the available locations, run the [az appservice list-locations](#) command. You generally create resources in a region near you.

4. Create an App Service plan with the [az appservice plan create](#) command.

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)

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
}
```

5. Create an app in the `myAppServicePlan` App Service plan with the [az webapp create](#) command.

The web app provides a hosting space for your API and provides a URL to view the deployed app.

In the following command, replace `<app_name>` with a unique name. If `<app_name>` is not unique, you get the error message "Website with given name `<app_name>` already exists." The default URL of the web app is `https://<app_name>.azurewebsites.net`.

```
az webapp create --name <app_name> --resource-group myResourceGroup --plan myAppServicePlan
```

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

```
{
  "availabilityState": "Normal",
  "clientAffinityEnabled": true,
  "clientCertEnabled": false,
  "cloningInfo": null,
  "containerSize": 0,
  "dailyMemoryTimeQuota": 0,
  "defaultHostName": "<app_name>.azurewebsites.net",
  "enabled": true,
  "enabledHostNames": [
    "<app_name>.azurewebsites.net",
    "<app_name>.scm.azurewebsites.net"
  ],
  "gatewaySiteName": null,
  "hostNameSslStates": [
    {
      "hostType": "Standard",
      "name": "<app_name>.azurewebsites.net",
      "sslState": "Disabled",
      "thumbprint": null,
      "toUpdate": null,
      "virtualIp": null
    }
  ],
  < JSON data removed for brevity. >
}
```

Deploy the API with Git

Deploy your code to the API app by pushing commits from your local Git repository to Azure App Service.

1. In the Cloud Shell, create deployment credentials with the [az webapp deployment user set](#) command. A deployment user is required for FTP and local Git deployment to a web app. The user name and password

are account level. *They are different from your Azure subscription credentials.*

In the following example, replace <username> and <password> (including brackets) with a new user name and password. The user name must be unique. The password must be at least eight characters long, with two of the following three elements: letters, numbers, symbols.

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

If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

You create this deployment user only once; you can use it for all your Azure deployments.

NOTE

Record the user name and password. You need them to deploy the web app later.

2. Initialize a new repo in the *ContactList* directory.

```
git init .
```

3. Exclude the *node_modules* directory created by npm in an earlier step in the tutorial from Git. Create a new `.gitignore` file in the current directory and add the following text on a new line anywhere in the file.

```
node_modules/
```

Confirm the `node_modules` folder is being ignored with `git status`.

4. Commit the changes to the repo.

```
git add .
git commit -m "initial version"
```

5. Use the Azure CLI to get the remote deployment URL for your API App. In the following command, replace <app_name> with your web app's name.

```
az webapp deployment source config-local-git --name <app_name> --resource-group myResourceGroup --query url --output tsv
```

Configure your local Git deployment to be able to push to the remote.

```
git remote add azure <URI from previous step>
```

Push to the Azure remote to deploy your app. You are prompted for the password you created earlier when you created the deployment user. Make sure that you enter the password you created in earlier in the quickstart, and not the password you use to log in to the Azure portal.

```
git push azure master
```

Test the API in Azure

1. Open a browser to http://app_name.azurewebsites.net/contacts. You see the same JSON returned as when you made the request locally earlier in the tutorial.

```
{  
    "id": 1,  
    "name": "Barney Poland",  
    "email": "barney@contoso.com"  
},  
{  
    "id": 2,  
    "name": "Lacy Barrera",  
    "email": "lacy@contoso.com"  
},  
{  
    "id": 3,  
    "name": "Lora Riggs",  
    "email": "lora@contoso.com"  
}
```

2. In a browser, go to the http://app_name.azurewebsites.net/docs endpoint to try out the Swagger UI running on Azure.

The screenshot shows a browser window with the title 'Swagger UI'. The address bar contains the URL <http://contactsapp.azurewebsites.net/docs?url=%2Fswagger#/Contacts>. The main content area is titled 'Contact List' and describes it as 'A Contact list API based on Swagger and built using Node.js'. Below this, there is a section titled 'Contacts' with two 'GET' operations listed: '/contacts' and '/contacts/{id}'. At the bottom left, it says '[BASE URL: / , API VERSION: v1]'. On the right side, there are buttons for 'Show/Hide', 'List Operations', 'Expand Operations', 'VALID', and a copy icon. The entire interface is styled with a green header and light gray background.

You can now deploy updates to the sample API to Azure simply by pushing commits to the Azure Git repository.

Clean up

To clean up the resources created in this quickstart, run the following Azure CLI command:

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

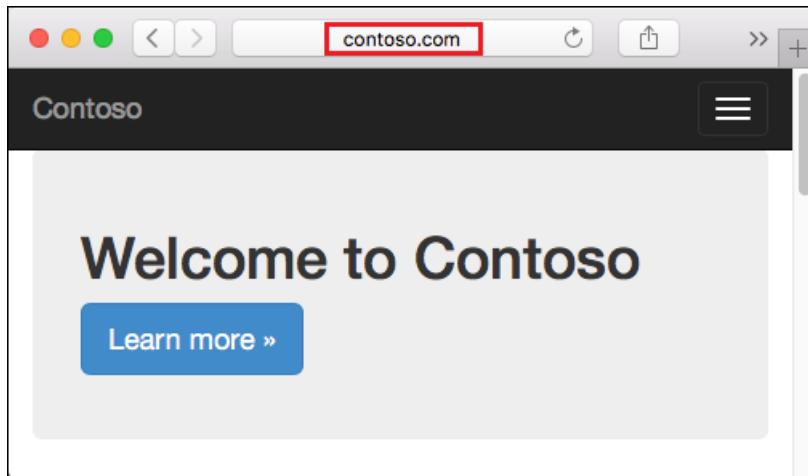
Next step

[Map an existing custom DNS name to Azure Web Apps](#)

Map an existing custom DNS name to Azure Web Apps

1/2/2018 • 10 min to read • [Edit Online](#)

Azure Web Apps provides a highly scalable, self-patching web hosting service. This tutorial shows you how to map an existing custom DNS name to Azure Web Apps.



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
- Automate domain mapping with scripts

You can use either a **CNAME record** or an **A record** to map a custom DNS name to App Service.

NOTE

We recommend that you use a CNAME for all custom DNS names except a root domain (for example, `contoso.com`).

To migrate a live site and its DNS domain name to App Service, see [Migrate an active DNS name to Azure App Service](#).

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**, or **Premium**). In this step, you make sure that the App Service app is in the supported pricing tier.

NOTE

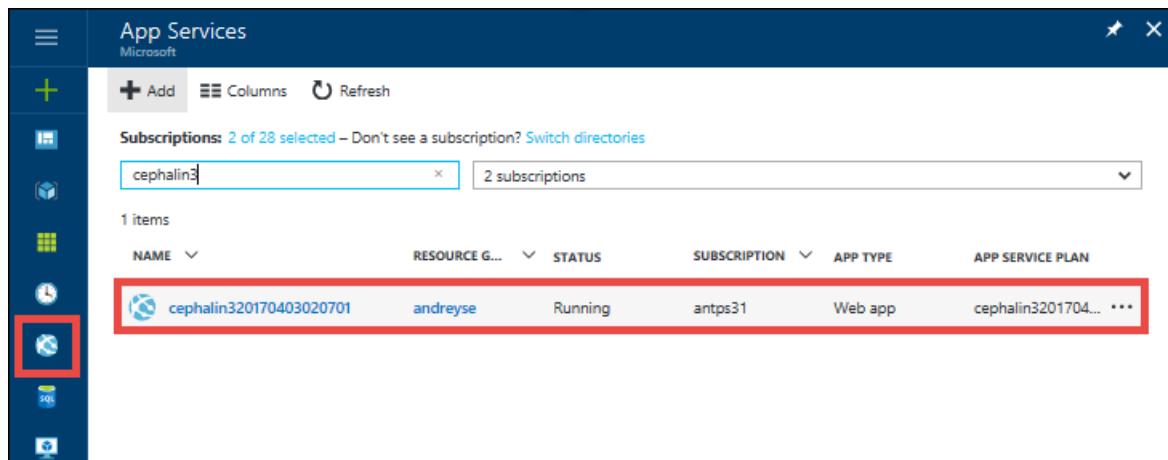
App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure VM as other App Service apps. Some apps may belong to other customers. These tiers are intended to be used only for development and testing purposes.

Sign in to Azure

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

Navigate to the app in the Azure portal

From the left menu, select **App Services**, and then select the name of the app.



The screenshot shows the Azure App Services management interface. On the left is a navigation bar with icons for Home, App Services, Functions, Logic Apps, Container Registry, and App Configuration. The 'App Services' icon is highlighted with a red box. The main area is titled 'App Services Microsoft' and shows a list of apps. A search bar at the top says 'Subscriptions: 2 of 28 selected - Don't see a subscription? Switch directories' and has a dropdown set to '2 subscriptions'. Below is a table with columns: NAME, RESOURCE G..., STATUS, SUBSCRIPTION, APP TYPE, and APP SERVICE PLAN. One row is selected and highlighted with a red border: 'cephalin320170403020701' (NAME), 'andreyse' (OWNER), 'Running' (STATUS), 'antps31' (SUBSCRIPTION), 'Web app' (APP TYPE), and 'cephalin3201704...' (APP SERVICE PLAN). The 'NAME' column is sorted in descending order.

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)**.



The screenshot shows the 'cephalin320170403020701' app service settings page. The left sidebar has several options: 'SSL certificates', 'Networking', 'Scale up (App Service plan)' (which is highlighted with a red box), 'Scale out (App Service plan)', 'Security scanning', and 'WebJobs'. The main content area is currently empty.

The app's current tier is highlighted by a blue border. Check to make sure that the app is not in the **Free** tier. Custom DNS is not supported in the **Free** tier.

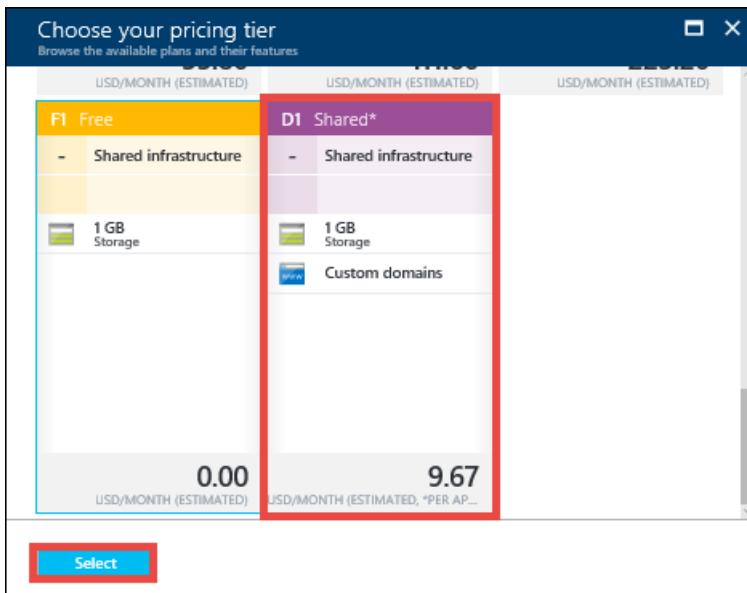
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)
F1 Free	D1 Shared*
- Shared infrastructure	- Shared infrastructure
 1 GB Storage	 1 GB Storage
	 Custom domains
0.00	9.67
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED, *PER AP...)
Select	

If the App Service plan is not **Free**, close the **Choose your pricing tier** page and skip to [Map a CNAME record](#).

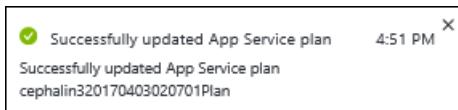
Scale up the App Service plan

Select any of the non-free tiers (**Shared, Basic, Standard, or Premium**).

Click **Select**.



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



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

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:

Last updated 4/4/2017 12:46 PM

Type	Name	Value	TTL
NS	@	ns07.domaincontrol.com	1 Hour
NS	@	ns08.domaincontrol.com	1 Hour

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 hostname (`<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:

Last updated 4/4/2017 3:45 PM

Type	Name	Value	TTL
CNAME	www	cephalin320170403020701.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**.

cephalin320170403020701
App Service

- Search (Ctrl+ /)
- Application settings
- Authentication / Authorization
- Backups
- Custom domains**
- SSL certificates
- Networking

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 hostname**.

The screenshot shows the 'Custom domains' section of an Azure portal. It includes a table of purchased domains (cephalinssl.biz and cephalincontoso.com), an external IP address (13.84.46.29), and a list of hostnames assigned to the site (cephalin320170403020701.azurewebsites.net). A red box highlights the 'Add hostname' button.

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

Select **Validate**.

The **Add hostname** button is activated.

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

Select **Add hostname**.

The screenshot shows the 'Add hostname' configuration dialog. It includes fields for the hostname (www.contoso.com), validation status (green checkmark), hostname record type (CNAME (www.example.com or any subdomain)), and a CNAME configuration link. A red box highlights the 'Add hostname' 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 'HOSTNAMES ASSIGNED TO SITE' list, which now includes the newly added hostname 'www.contoso.com'. A red box highlights the list item.

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

Domain Verification

Domain ownership ! Error

Hostname availability ✓ OK

No CNAME records were found. Please add a CNAME record pointing to cephalin320170403020701.azurewebsites.net

Map an A record

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

Copy the app's IP address

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

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

cephalin320170403020701 App Service

Search (Ctrl+ /)

Application settings

Authentication / Authorization

Backups

Custom domains

SSL certificates

Networking

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

Purchased Domains

DOMAINS	EXPIRES	STATUS	...
cephalinssl.biz	2018-03-16	Active	...

External IP Address

Use this IP Address to configure your DNS settings for A records

IP Address: `13.84.46.29`

Hostnames

Add hostname

HOSTNAMES ASSIGNED TO SITE

cephalin320170403020701.azurewebsites.net

Access DNS records with domain provider

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:

Last updated 4/4/2017 12:46 PM

Type	Name	Value	TTL
NS	@	ns07.domaincontrol.com	1 Hour
NS	@	ns08.domaincontrol.com	1 Hour

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 hostname <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	@	<app_name>.azurewebsites.net

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

Last updated 4/4/2017 3:40 PM

Type	Name	Value	TTL	
A	@	13.84.46.29	1 Hour	
TXT	@	cephalin320170403020701.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 hostname**.

The screenshot shows the 'Purchased Domains' section with one entry: 'cephalinssl.biz' (Expires: 2018-03-16, Status: Active). Below it is the 'External IP Address' section, which includes a note to use the IP address for A records and a text input field containing '13.84.46.29'. Under 'Hostnames', there is an 'Add hostname' button and a list titled 'HOSTNAMES ASSIGNED TO SITE' containing 'cephalin320170403020701.azurewebsites.net'.

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

Select **Validate**.

The **Add hostname** button is activated.

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

Select **Add hostname**.

The dialog box has a title 'Add hostname' and a sub-header 'cephalin320170403020701'. It contains two main sections: 'Hostname' (containing 'contoso.com') and 'Hostname record type' (set to 'A record (example.com)'). Below these are descriptive text and links ('A record configuration', 'Learn More'), an 'External IP Address' input field ('13.84.46.29'), and an 'Add hostname' 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 list shows the newly added 'contoso.com' entry along with the previous entry 'cephalin320170403020701.azurewebsites.net'.

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

The screenshot shows a 'Domain Verification' section. It has two status indicators: 'Domain ownership' with an 'Error' icon and 'Hostname availability' with an 'OK' icon. Below these, a red exclamation mark icon is displayed. A message box contains two error messages: 'No A records were found. Please create an A record pointing to the following IP address: 13.84.46.29' and 'No TXT records were found. Please create a TXT record pointing from contoso.com to cephalin320170403020701.azurewebsites.net'.

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

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:

The screenshot shows a table titled 'Records'. It includes a timestamp 'Last updated 4/4/2017 12:46 PM'. The table has columns: Type, Name, Value, and TTL. There are two entries: 'NS' with 'Name' '@' and 'Value' 'ns07.domaincontrol.com' with 'TTL' '1 Hour'; and another 'NS' entry with 'Name' '@' and 'Value' 'ns08.domaincontrol.com' with 'TTL' '1 Hour'. At the bottom right is a blue 'ADD' button.

Type	Name	Value	TTL
NS	@	ns07.domaincontrol.com	1 Hour
NS	@	ns08.domaincontrol.com	1 Hour

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 hostname (`<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:

Records

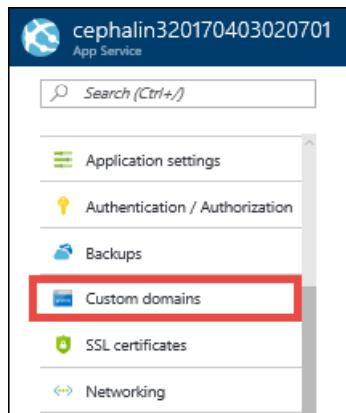
Last updated 4/4/2017 3:45 PM

Type	Name	Value	TTL	
CNAME	*	cephalin320170403020701.azurewebsites.net	1 Hour	
				ADD

Enable the CNAME record mapping in the app

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

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



Select the + icon next to **Add hostname**.

A screenshot of the 'Custom domains' blade in the Azure portal. It shows a table of purchased domains and an external IP address. The 'Add hostname' button is highlighted with a red box.

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

The **Add hostname** button is activated.

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

Select **Add hostname**.

The screenshot shows the 'Add Hostname' configuration page. At the top, there is a field labeled 'Hostname' containing 'sub1.contoso.com' with a green checkmark icon. Below it is a blue 'Validate' button. A section titled 'Hostname record type' has a dropdown menu set to 'CNAME (www.example.com or any subdomain)'. Underneath, there is a globe icon and a link to 'CNAME configuration'. A note states: 'A CNAME record is used to specify that a domain name is an alias for another domain.' with a 'Learn More' link. A 'CNAME' field contains 'cephalin320170403020701.azurewebsites.net'. At the bottom is a blue 'Add hostname' 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.

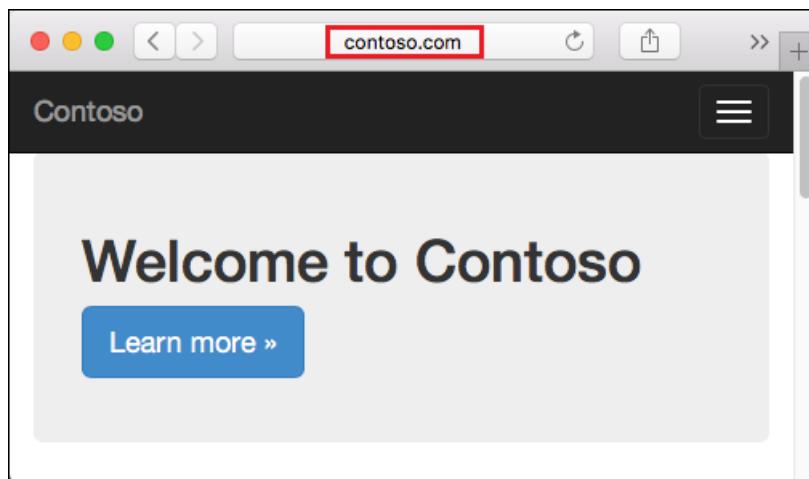
Select the + icon again to add another hostname that matches the wildcard domain. For example, add

`sub2.contoso.com`.

HOSTNAMES ASSIGNED TO SITE	
sub2.contoso.com	...
sub1.contoso.com	...
cephalin320170403020701.azurewebsites.net	

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 error "Web Site 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`.

Direct default URL to a custom directory

By default, App Service directs web requests to the root directory of your app code. However, certain web frameworks don't start in the root directory. For example, [Laravel](#) starts in the `public` subdirectory. To continue the `contoso.com` DNS example, such an app would be accessible at `http://contoso.com/public`, but you would really want to direct `http://contoso.com` to the `public` directory instead. This step doesn't involve DNS resolution, but customizing the virtual directory.

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

At the bottom of the page, the root virtual directory `/` points to `site\wwwroot` by default, which is the root directory of your app code. Change it to point to the `site\wwwroot\public` instead, for example, and save your changes.

The screenshot shows the 'Application settings' page for an Azure App Service. The left sidebar lists various settings: Application settings (selected), Authentication / Authorization, Managed service identity, Backups, Custom domains, SSL certificates, Networking, and Scale up (App Service plan). The main area shows 'Handler mappings' (No results) and 'Virtual applications and directories'. Under 'Virtual applications and directories', there is a row for the root path '/'. The 'Virtual directory' field contains '/' and the 'Physical path relative to site' field contains 'site\wwwroot\public' with a green checkmark. A red box highlights the 'Physical path relative to site' field. At the top right, there are 'Save' and 'Discard' buttons, with 'Save' being highlighted by a red box.

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

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

```
Set-AzureRmWebApp ` 
-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
- Automate domain mapping with scripts

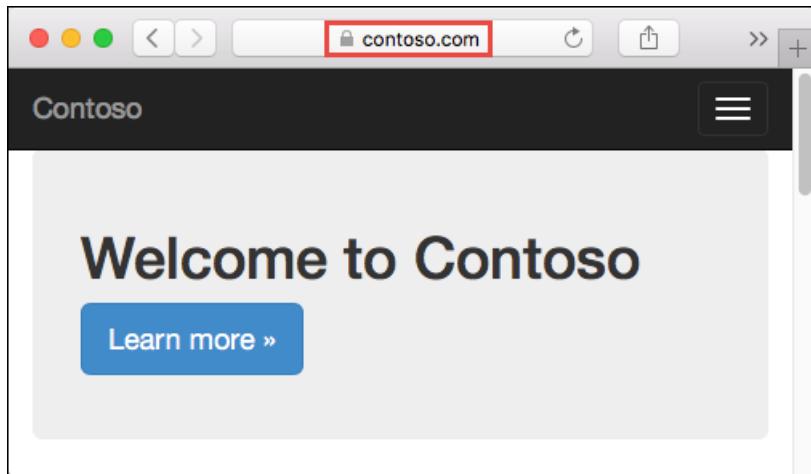
Advance to the next tutorial to learn how to bind a custom SSL certificate to a web app.

[Bind an existing custom SSL certificate to Azure Web Apps](#)

Bind an existing custom SSL certificate to Azure Web Apps

12/1/2017 • 7 min to read • [Edit Online](#)

Azure Web Apps provides a highly scalable, self-patching web hosting service. This tutorial shows you how to bind a custom SSL certificate that you purchased from a trusted certificate authority to [Azure Web Apps](#). When you're finished, you'll be able to access your web app at the HTTPS endpoint of your custom DNS domain.



In this tutorial, you learn how to:

- Upgrade your app's pricing tier
- Bind your custom SSL certificate to App Service
- Enforce HTTPS for your app
- Automate SSL certificate binding with scripts

NOTE

If you need to get a custom SSL certificate, you can get one in the Azure portal directly and bind it to your web app. Follow the [App Service Certificates tutorial](#).

Prerequisites

To complete this tutorial:

- [Create an App Service app](#)
- [Map a custom DNS name to your web app](#)
- Acquire an SSL certificate from a trusted certificate authority
- Have the private key you used to sign the SSL certificate request

Requirements for your SSL certificate

To use a certificate in App Service, the certificate must meet all the following requirements:

- Signed by a trusted certificate authority
- Exported as a password-protected PFX file
- Contains private key at least 2048 bits long
- Contains all intermediate certificates in the certificate chain

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

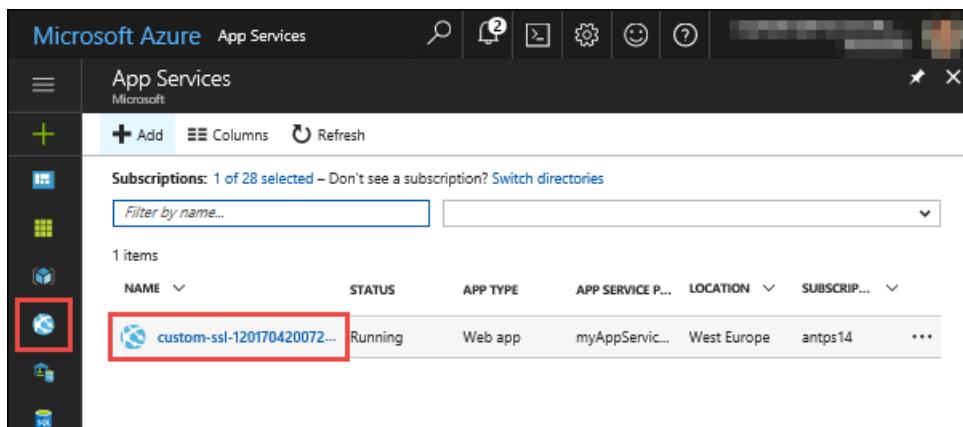
To bind a custom SSL certificate to your web app, your [App Service plan](#) must be in the **Basic**, **Standard**, or **Premium** tier. In this step, you make sure that your web app is in the supported pricing tier.

Log in to Azure

Open the [Azure portal](#).

Navigate to your web app

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

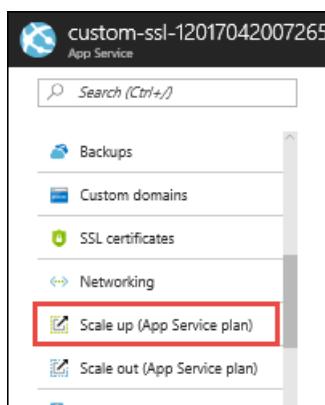


NAME	STATUS	APP TYPE	APP SERVICE P...	LOCATION	SUBSCRIP...
custom-ssl-120170420072...	Running	Web app	myAppService...	West Europe	antps14

You have landed in 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 **Free** or **Shared** tier. Your web app's current tier is highlighted by a dark blue box.

F1 Free	D1 Shared*
- Shared infrastructure	- Shared infrastructure
1 GB Storage	1 GB Storage
0.00 USD/MONTH (ESTIMATED)	9.67 USD/MONTH (ESTIMATED) *PER AP...
Select	

Custom SSL is not supported in the **Free** or **Shared** tier. If you need to scale up, follow the steps in the next section. Otherwise, close the **Choose your pricing tier** page and skip to [Upload and bind your SSL certificate](#).

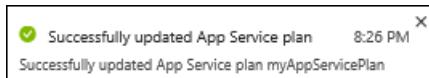
Scale up your App Service plan

Select one of the **Basic**, **Standard**, or **Premium** tiers.

Click **Select**.

Choose your pricing tier		
Browse the available plans and their features		
PRICES	PRICES	PRICES
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)
B1 Basic	B2 Basic	B3 Basic
1 Core	2 Core	4 Core
1.75 GB RAM	3.5 GB RAM	7 GB RAM
10 GB Storage	10 GB Storage	10 GB Storage
Custom domains	Custom domains	Custom domains
SSL Support SNI SSL Included	SSL Support SNI SSL Included	SSL Support SNI SSL Included
Up to 3 instances Manual scale	Up to 3 instances Manual scale	Up to 3 instances Manual scale
55.80 USD/MONTH (ESTIMATED)	111.60 USD/MONTH (ESTIMATED)	223.20 USD/MONTH (ESTIMATED)
F1 Free	D1 Shared*	
- Shared infrastructure	- Shared infrastructure	
Select		

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



Bind your SSL certificate

You are ready to upload your SSL certificate to your web app.

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 your SSL certificate

To upload your SSL certificate, click **SSL certificates** in the left navigation of your web app.

Click **Upload Certificate**.

In **PFX Certificate File**, select your PFX file. In **Certificate password**, type the password that you created when you exported the PFX file.

Click **Upload**.

Add certificate X

i Secure your app's custom domain with HTTPS ?

Type Private Public

* PFX Certificate File
 Select a file Browse

Certificate password

When App Service finishes uploading your certificate, it appears in the **SSL certificates** page.

The screenshot shows the 'SSL certificates' page. At the top, there is a green shield icon with a lock and the word 'SSL'. Below it, a message says: 'Certificates must be associated with your application before you can use them to create a binding. You can upload a certificate you purchased externally, or import an App Service Certificate.' There are two buttons: 'Import App Service Certificate' with a download arrow icon and 'Upload Certificate' with an upload arrow icon. A table lists uploaded certificates with columns: NAME, EXPIRATION, and THUMBPRINT. One row is highlighted with a red box around the entire row, showing 'contoso.com,www.c...' in the NAME column, '4/20/2018' in the EXPIRATION column, and a blurred thumbprint in the THUMBPRINT column. Below the table, under 'SSL bindings', there is a button 'Add binding' with a plus sign icon. At the bottom, a note says 'No results'.

Bind your SSL certificate

In the **SSL bindings** section, click **Add binding**.

In the **Add SSL Binding** page, use the dropdowns to select the domain name to secure, and the certificate to use.

NOTE

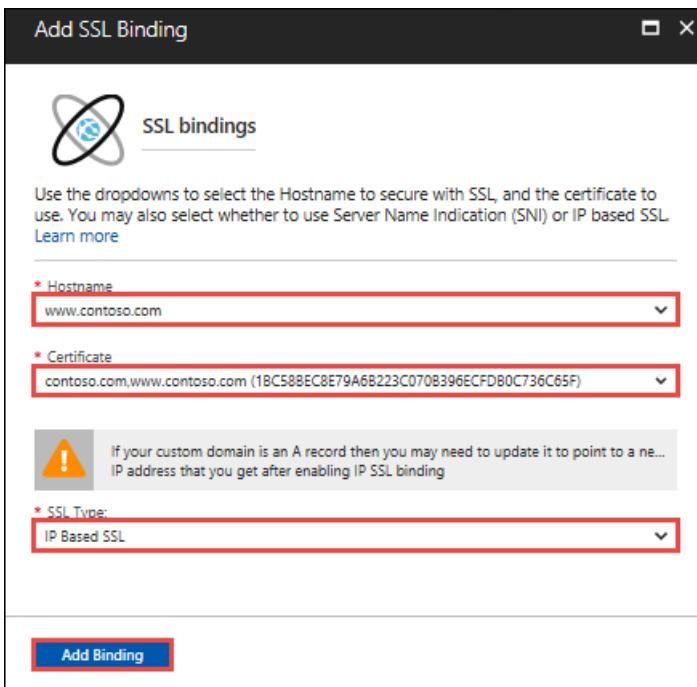
If you have uploaded your certificate but don't see the domain name(s) in the **Hostname** dropdown, try refreshing the browser page.

In **SSL Type**, select whether to use **Server Name Indication (SNI)** or IP-based SSL.

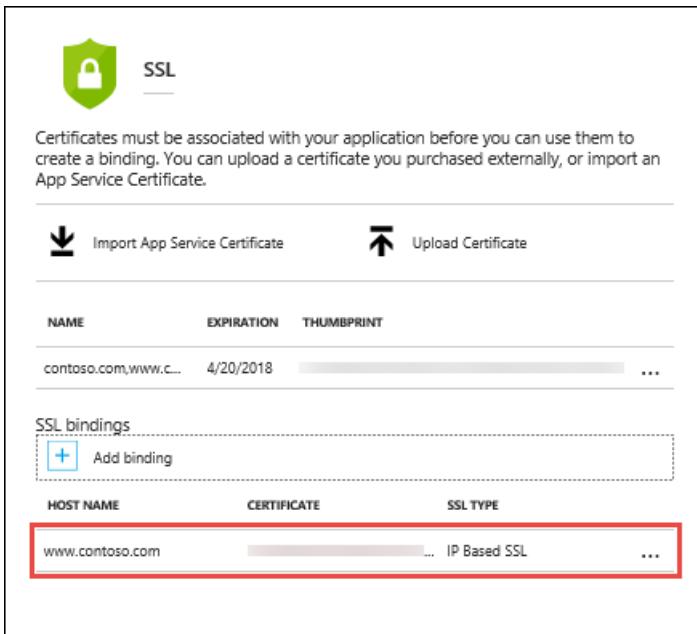
- **SNI-based SSL** - Multiple SNI-based SSL bindings may be added. This option allows multiple SSL certificates to secure multiple domains on the same IP address. Most modern browsers (including Internet Explorer, Chrome, Firefox, and Opera) support SNI (find more comprehensive browser support information at [Server Name Indication](#)).

- **IP-based SSL** - Only one IP-based SSL binding may be added. This option allows only one SSL certificate to secure a dedicated public IP address. To secure multiple domains, you must secure them all using the same SSL certificate. This is the traditional option for SSL binding.

Click **Add Binding**.



When App Service finishes uploading your certificate, it appears in the **SSL bindings** section.



Remap A record for IP SSL

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

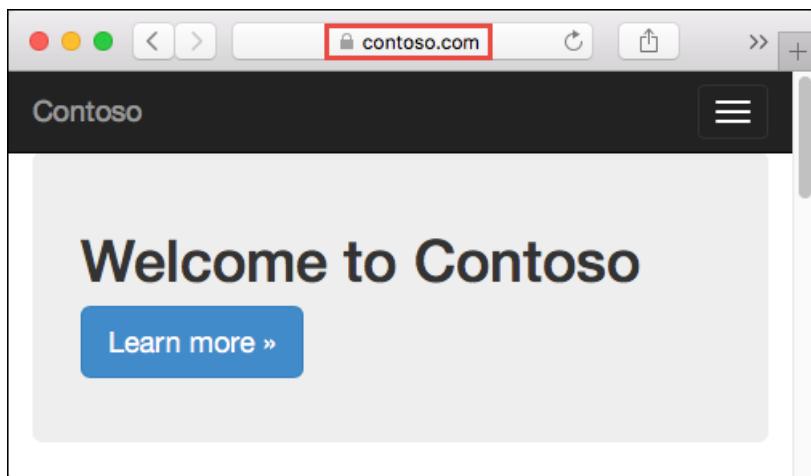
By default, your web app uses a shared public IP address. When you bind a certificate with IP-based SSL, App Service creates a new, dedicated IP address for your web app.

If you have mapped an A record to your web app, update your domain registry with this new, dedicated IP address.

Your web 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

All that's left to do now is to make sure that HTTPS works for your custom domain. In various browsers, browse to `https://<your.custom.domain>` to see that it serves up your web app.



NOTE

If your web 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.

Enforce HTTPS

By default, anyone can still access your web app using HTTP. You can redirect all HTTP requests to the HTTPS port.

In your web app page, in the left navigation, select **Custom domains**. Then, in **HTTPS Only**, select **On**.

A screenshot of the Azure portal's 'Custom domains' settings page. The left sidebar shows a list of settings: Application settings, Authentication / Authorization, Managed service identity, Backups, Custom domains (which is selected and highlighted with a red box), SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs, Push, and MySQL In App. The main panel is titled 'Custom Hostnames' and contains instructions to 'Configure and manage custom hostnames assigned to your app'. It shows the IP address as 52.174.27.189 and the 'HTTPS Only' toggle switch set to 'On' (also highlighted with a red box). Below this, there's a 'Add hostname' button and a list of assigned hostnames: contoso.com, www.contoso.com, and custom-ssl-120170420072657.azurewebsites.net.

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>

Automate with scripts

You can automate SSL bindings for your web app with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

Azure CLI

The following command uploads an exported PFX file and gets the thumbprint.

```
thumbprint=$(az webapp config ssl upload \
    --name <app_name> \
    --resource-group <resource_group_name> \
    --certificate-file <path_to_PFX_file> \
    --certificate-password <PFX_password> \
    --query thumbprint \
    --output tsv)
```

The following command adds an SNI-based SSL binding, using the thumbprint from the previous command.

```
az webapp config ssl bind \
    --name <app_name> \
    --resource-group <resource_group_name>
    --certificate-thumbprint $thumbprint \
    --ssl-type SNI \
```

Azure PowerShell

The following command uploads an exported PFX file and adds an SNI-based SSL binding.

```
New-AzureRmWebAppSSLBinding ` 
    -WebAppName <app_name> ` 
    -ResourceGroupName <resource_group_name> ` 
    -Name <dns_name> ` 
    -CertificateFilePath <path_to_PFX_file> ` 
    -CertificatePassword <PFX_password> ` 
    -SslState SniEnabled
```

Public certificates (optional)

You can upload [public certificates](#) to your web app. You can use public certificates for apps in App Service Environments also. If you need to store the certificate in the LocalMachine certificate store, you need to use a web app on App Service Environment. For more information, see [How to configure public certificates to your Web App](#).

Add certificate X

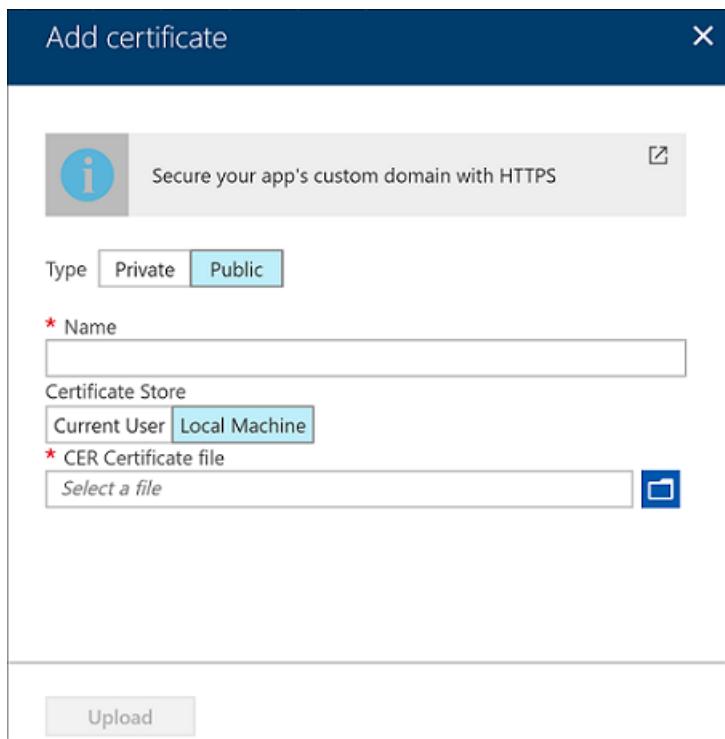
Secure your app's custom domain with HTTPS [Close]

Type Private Public

* Name

Certificate Store Current User Local Machine

* CER Certificate file Select a file [Browse]



Next steps

In this tutorial, you learned how to:

- Upgrade your app's pricing tier
- Bind your custom SSL certificate to App Service
- Enforce HTTPS for your app
- Automate SSL certificate binding with scripts

Advance to the next tutorial to learn how to use Azure Content Delivery Network.

[Add a Content Delivery Network \(CDN\) to an Azure App Service](#)

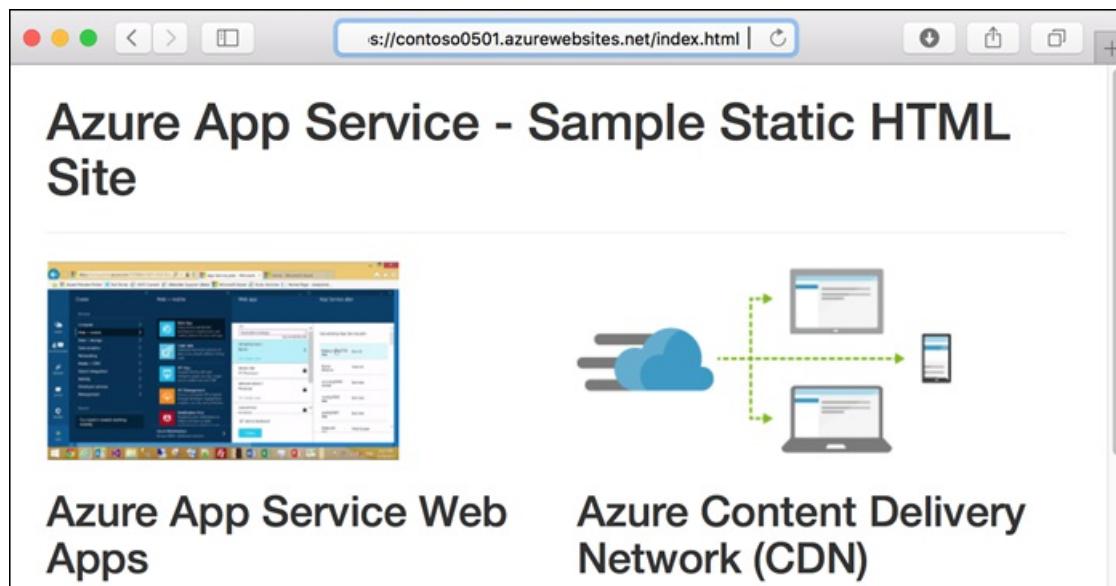
For more information, see [Use an SSL certificate in your application code in Azure App Service](#).

Add a Content Delivery Network (CDN) to an Azure App Service

9/19/2017 • 8 min to read • [Edit Online](#)

Azure Content Delivery Network (CDN) caches static web content at strategically placed locations to provide maximum throughput for delivering content to users. The CDN also decreases server load on your web app. This tutorial shows how to add Azure CDN to a [web app in Azure App Service](#).

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.
- Use a custom domain for the CDN endpoint.

Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install Azure CLI 2.0](#)

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.

Have a custom domain ready

To complete the custom domain step of this tutorial, you need to own a custom domain and have access to your DNS registry for your domain provider (such as GoDaddy). For example, to add DNS entries for `contoso.com` and `www.contoso.com`, you must have access to configure the DNS settings for the `contoso.com` root domain.

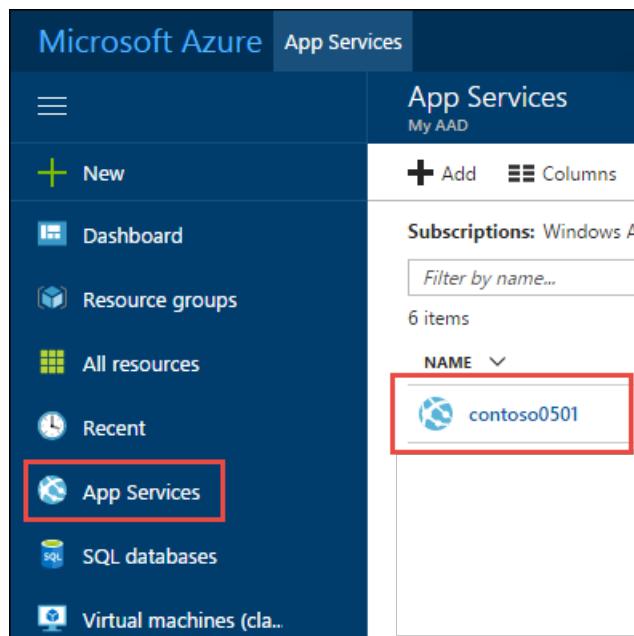
If you don't already have a domain name, consider following the [App Service domain tutorial](#) to purchase a domain using the Azure portal.

Log in to the Azure portal

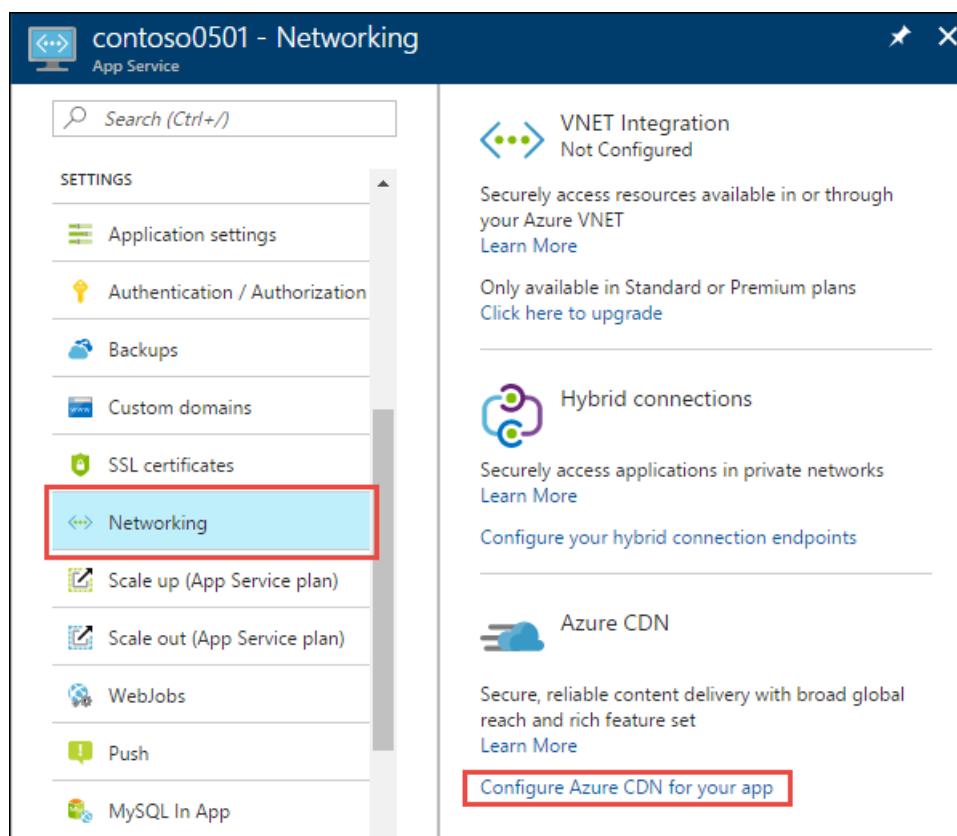
Open a browser and navigate to the [Azure portal](#).

Create a CDN profile and endpoint

In the left navigation, select **App Services**, and then select the app that you created in the [static HTML quickstart](#).



In the **App Service** page, in the **Settings** section, select **Networking > Configure Azure CDN for your app**.



In the **Azure Content Delivery Network** page, provide the **New endpoint** settings as specified in the table.

Azure CDN



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	Select Create new to create a CDN profile. 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. In this tutorial, we are using Standard Akamai.
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**.

Azure creates the profile and endpoint. The new endpoint appears in the **Endpoints** list on the same page, 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

If you selected Verizon pricing tier, it typically takes about 90 minutes for endpoint propagation. For Akamai, it takes a couple minutes for propagation

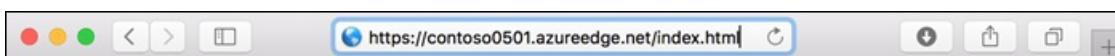
The sample app has an `index.html` file and `css`, `img`, and `js` folders that contain other static assets. The content paths for all of these files are the same at the CDN endpoint. For example, both of the following URLs access the `bootstrap.css` file in the `css` folder:

```
http://<appname>.azurewebsites.net/css/bootstrap.css
```

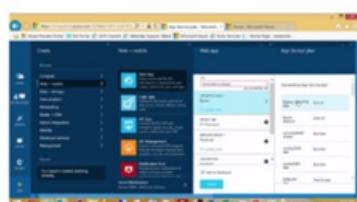
```
http://<endpointname>.azureedge.net/css/bootstrap.css
```

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, you can 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 and you see the change.

```
http://<appname>.azurewebsites.net/index.html
```



Browse to the CDN endpoint URL for the home page and you don't see the change because the cached version in the CDN hasn't expired yet.

```
http://<endpointname>.azureedge.net/index.html
```



Purge the CDN in the portal

To trigger the CDN to update its cached version, purge the CDN.

In the portal left navigation, select **Resource groups**, and then select the resource group that you created for your web app (`myResourceGroup`).

The screenshot shows the Microsoft Azure Resource groups page. On the left sidebar, under the 'Resource groups' category, there is a box labeled 'myResourceGroup' which is highlighted with a red border. The main area displays a table titled 'Resource groups' with one item listed:

NAME
myResourceGroup

In the list of resources, select your CDN endpoint.

The screenshot shows the 'myResourceGroup' resource group details page. On the left sidebar, under the 'SETTINGS' category, there is a box labeled 'contoso0501' which is highlighted with a red border. The main area displays a table titled 'Essentials' with one item listed:

NAME	TYPE
contoso0501	Endpoint

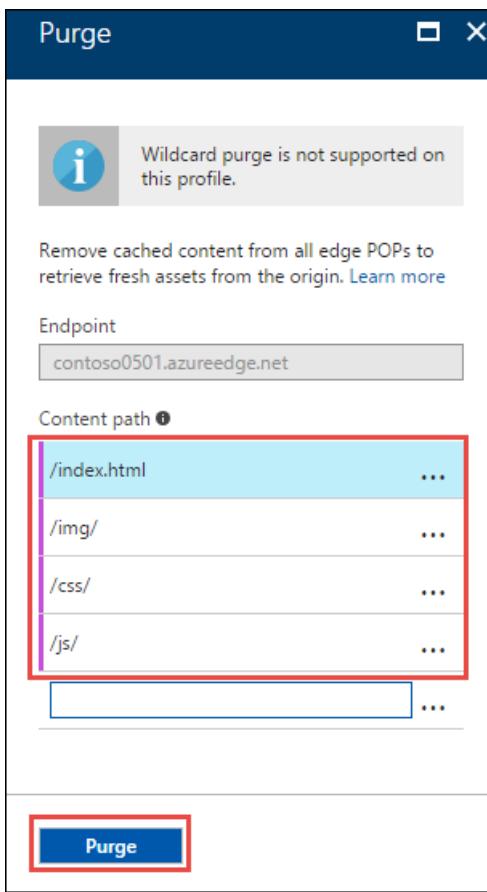
At the top of the **Endpoint** page, click **Purge**.

The screenshot shows the 'contoso0501' endpoint page. At the top right, there is a red-bordered button labeled 'Purge'. The main area displays a table titled 'Essentials' with one item listed:

Resource group
myResourceGroup

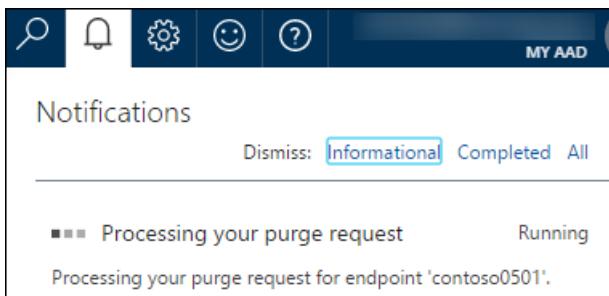
Enter the content paths you wish to purge. You can pass a complete file path to purge an individual file, or a path segment to purge and refresh all content in a folder. Since you changed `index.html`, make sure that is one of the paths.

At the bottom of the page, select **Purge**.



Verify that the CDN is updated

Wait until the purge request finishes processing, typically a couple of minutes. To see the current status, select the bell icon at the top of the page.



Browse to the CDN endpoint URL for `index.html`, and now you see the V2 that you added to the title on the home page. This shows that the CDN cache has been refreshed.

The screenshot shows a web browser window with the URL `https://contoso0501.azureedge.net/index.html` in the address bar. The page content is 'Azure App Service - Sample Static HTML Site - V2', which is different from the previous version shown in the earlier screenshots. The browser interface includes standard navigation buttons and a tab bar.

For more information, see [Purge an Azure CDN endpoint](#).

Use query strings to version content

The Azure CDN offers the following caching behavior options:

- Ignore query strings

- Bypass caching for query strings
- Cache every unique URL

The first of these is the default, which means there is only one cached version of an asset regardless of the query string in the URL.

In this section of the tutorial, you change the caching behavior to cache every unique URL.

Change the cache behavior

In the Azure portal **CDN Endpoint** page, select **Cache**.

Select **Cache every unique URL** from the **Query string caching behavior** drop-down list.

Select **Save**.

contoso0501 - Cache
Endpoint

Search (Ctrl+ /)

Save **Discard**

About This Feature
Decide how CDN caches requests that include query strings. You can ignore any query contain query strings from being cached, or cache every request with a unique URL.
[Learn more](#)

Configure

Query string caching behavior: **Cache every unique URL**

Ignore query strings
Ignore query strings
Bypass caching for query strings
Cache every unique URL

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems

SETTINGS

Origin
Custom domains
Compression
Cache

Verify that unique URLs are cached separately

In a browser, navigate to the home page at the CDN endpoint, but include a query string:

```
http://<endpointname>.azureedge.net/index.html?q=1
```

The 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` and change "V2" to "V3", and 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`. The CDN gets the current `index.html` file and displays "V3". But 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](#).

Map a custom domain to a CDN endpoint

You'll map your custom domain to your CDN Endpoint by creating a CNAME record. A CNAME record is a DNS feature that maps a source domain to a destination domain. For example, you might map `cdn.contoso.com` or `static.contoso.com` to `contoso.azureedge.net`.

If you don't have a custom domain, consider following the [App Service domain tutorial](#) to purchase a domain using the Azure portal.

Find the hostname to use with the CNAME

In the Azure portal **Endpoint** page, make sure **Overview** is selected in the left navigation, and then select the **+ Custom Domain** button at the top of the page.

The screenshot shows the Azure CDN endpoint management interface. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Origin, Custom domains, and Compression. The main area has tabs for Essentials, Custom domains, Purge, Load, and Stop. The 'Custom domains' tab is selected and highlighted with a red box. Under 'Custom domains', it says 'HOSTNAME' and 'There are no custom domains to display'. Other sections include Resource group (myResourceGroup), Status (Running), Location (West Europe), Subscription name (Windows Azure MSDN - Visual Studio Ultimate), and Subscription ID.

In the **Add a custom domain** page, you see the endpoint host name to use in creating a CNAME record. The host name is derived from your CDN endpoint URL: <EndpointName>.azureedge.net.

The screenshot shows the 'Add a custom domain' dialog box. It has a title bar with 'Add a custom domain' and close (X) and minimize (□) buttons. The main content area contains instructions: 'Create a DNS mapping from your custom hostname to the CDN endpoint hostname with your DNS provider. Then type the custom hostname here for verification.' Below this is a 'Learn more' link. There are two input fields: 'Endpoint hostname' containing 'contoso0501.azureedge.net' and 'Custom hostname' with a placeholder field. At the bottom are 'Add' and 'Automation options' buttons.

Configure the CNAME with your domain registrar

Navigate to your domain registrar's web site, and locate the section for creating DNS records. You might find this in a section such as **Domain Name, DNS, or Name Server Management**.

Find the section for managing CNAMEs. You may have to go to an advanced settings page and look for the words CNAME, Alias, or Subdomains.

Create a CNAME record that maps your chosen subdomain (for example, **static** or **cdn**) to the **Endpoint host name** shown earlier in the portal.

Enter the custom domain in Azure

Return to the **Add a custom domain** page, and enter your custom domain, including the subdomain, in the dialog box. For example, enter `cdn.contoso.com`.

Azure verifies that the CNAME record exists for the domain name you have entered. If the CNAME is correct, your custom domain is validated.

It can take time for the CNAME record to propagate to name servers on the Internet. If your domain is not validated immediately, wait a few minutes and try again.

Test the custom domain

In a browser, navigate to the `index.html` file using your custom domain (for example, `cdn.contoso.com/index.html`) to verify that the result is the same as when you go directly to `<endpointname>azurededge.net/index.html`.



For more information, see [Map Azure CDN content to a custom domain](#).

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.
- Use a custom domain for the CDN endpoint.

Learn how to optimize CDN performance in the following articles:

[Improve performance by compressing files in Azure CDN](#)

[Pre-load assets on an Azure CDN endpoint](#)

Azure CLI Samples

12/12/2017 • 2 min to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure CLI.

Create app	
Create a web app and deploy files with FTP	Creates an Azure web app and deploys a file to it using FTP.
Create a web app and deploy code from GitHub	Creates an Azure web app and deploys code from a public GitHub repository.
Create a web app with continuous deployment from GitHub	Creates an Azure web app with continuous publishing from a GitHub repository you own.
Create a web app and deploy code from a local Git repository	Creates an Azure web app and configures code push from a local Git repository.
Create a web app and deploy code to a staging environment	Creates an Azure web app with a deployment slot for staging code changes.
Create an ASP.NET Core web app in a Docker container	Creates an Azure web app on Linux and loads a Docker image from Docker Hub.
Configure app	
Map a custom domain to a web app	Creates an Azure web app and maps a custom domain name to it.
Bind a custom SSL certificate to a web app	Creates an Azure web app and binds the SSL certificate of a custom domain name to it.
Scale app	
Scale a web app manually	Creates an Azure web app and scales it across 2 instances.
Scale a web app worldwide with a high-availability architecture	Creates two Azure web apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
Connect app to resources	
Connect a web app to a SQL Database	Creates an Azure web app and a SQL database, then adds the database connection string to the app settings.
Connect a web app to a storage account	Creates an Azure web app and a storage account, then adds the storage connection string to the app settings.
Connect a web app to a redis cache	Creates an Azure web app and a redis cache, then adds the redis connection details to the app settings.)

Connect a web app to Cosmos DB	Creates an Azure web app and a Cosmos DB, then adds the Cosmos DB connection details to the app settings.
Back up and restore app	
Back up a web app	Creates an Azure web app and creates a one-time backup for it.
Create a scheduled backup for a web app	Creates an Azure web app and creates a scheduled backup for it.
Restores a web app from a backup	Restores an Azure web app from a backup.
Monitor app	
Monitor a web app with web server logs	Creates an Azure web app, enables logging for it, and downloads the logs to your local machine.

Azure PowerShell Samples

12/4/2017 • 1 min to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure PowerShell.

Create app	
Create a web app with deployment from GitHub	Creates an Azure web app that pulls code from GitHub.
Create a web app with continuous deployment from GitHub	Creates an Azure web app that continuously deploys code from GitHub.
Create a web app and deploy code with FTP	Creates an Azure web app and upload files from a local directory using FTP.
Create a web app and deploy code from a local Git repository	Creates an Azure web app and configures code push from a local Git repository.
Create a web app and deploy code to a staging environment	Creates an Azure web app with a deployment slot for staging code changes.
Configure app	
Map a custom domain to a web app	Creates an Azure web app and maps a custom domain name to it.
Bind a custom SSL certificate to a web app	Creates an Azure web app and binds the SSL certificate of a custom domain name to it.
Scale app	
Scale a web app manually	Creates an Azure web app and scales it across 2 instances.
Scale a web app worldwide with a high-availability architecture	Creates two Azure web apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
Connect app to resources	
Connect a web app to a SQL Database	Creates an Azure web app and a SQL database, then adds the database connection string to the app settings.
Connect a web app to a storage account	Creates an Azure web app and a storage account, then adds the storage connection string to the app settings.
Back up and restore app	
Back up a web app	Creates an Azure web app and creates a one-time backup for it.

Create a scheduled backup for a web app	Creates an Azure web app and creates a scheduled backup for it.
Delete a backup for a web app	Deletes an existing backup for a web app.
Monitor app	
Monitor a web app with web server logs	Creates an Azure web app, enables logging for it, and downloads the logs to your local machine.

4 min to read •

Azure App Service plan overview

1/2/2018 • 7 min 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, Consumption)

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, which provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.
- **Consumption:** This tier is only available to [function apps](#). It scales the functions dynamically depending on workload. For more information, see [Azure Functions hosting plans comparison](#).

NOTE

App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure VM as other App Service apps. Some apps may 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.
- (Azure Functions only) The **Consumption** tier dynamically allocates VM instances to service a function app's workload, and is charged dynamically per second by Azure. For more information, see [Azure Functions pricing](#).

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 add a [custom SSL certificate](#), 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 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

1/2/2018 • 9 min to read • [Edit Online](#)

This article describes the common baseline operating system functionality that is available to all apps running on [Azure App Service](#). This functionality includes file, network, and registry access, and diagnostics logs and events.

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 VM as other App Service apps. Some apps may 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), and a "user" drive (the C:\ drive), whose size varies depending on the size of the VM.

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 very nicely to the common pattern of content storage used by on-premises web hosting environments that have multiple load-balanced servers.

Within App Service, there are 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.

Note that due to how cloud 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 cloud 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: <http://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. Note that 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 (e.g. 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.

More information

[Azure Web App 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.

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Authentication and authorization in Azure App Service

10/24/2017 • 9 min to read • [Edit Online](#)

What is App Service Authentication / Authorization?

App Service Authentication / Authorization is a feature that provides a way for your application to sign in users so that you don't have to change code on the app backend. It provides an easy way to protect your application and work with per-user data.

App Service uses federated identity, in which a third-party identity provider stores accounts and authenticates users. The application relies on the provider's identity information so that the app doesn't have to store that information itself. App Service supports five identity providers out of the box: Azure Active Directory, Facebook, Google, Microsoft Account, and Twitter. Your app can use any number of these identity providers to provide your users with options for how they sign in. To expand the built-in support, you can integrate another identity provider or [your own custom identity solution](#).

If you want to get started right away, see one of the following tutorials [Add authentication to your iOS app](#) (or [Android](#), [Windows](#), [Xamarin.iOS](#), [Xamarin.Android](#), [Xamarin.Forms](#), or [Cordova](#)).

How authentication works in App Service

In order to authenticate by using one of the identity providers, you first need to configure the identity provider to know about your application. The identity provider will then provide IDs and secrets that you provide to App Service. This completes the trust relationship so that App Service can validate user assertions, such as authentication tokens, from the identity provider.

To sign in a user by using one of these providers, the user must be redirected to an endpoint that signs in users for that provider. If customers are using a web browser, you can have App Service automatically direct all unauthenticated users to the endpoint that signs in users. Otherwise, you will need to direct your customers to `{your App Service base URL}/.auth/login/<provider>`, where `<provider>` is one of the following values: aad, facebook, google, microsoft, or twitter. Mobile and API scenarios are explained in sections later in this article.

Users who interact with your application through a web browser will have a cookie set so that they can remain authenticated as they browse your application. For other client types, such as mobile, a JSON web token (JWT), which should be presented in the `x-zumo-auth` header, will be issued to the client. The Mobile Apps client SDKs will handle this for you. Alternatively, an Azure Active Directory identity token or access token may be directly included in the `Authorization` header as a [bearer token](#).

App Service will validate any cookie or token that your application issues to authenticate users. To restrict who can access your application, see the [Authorization](#) section later in this article.

Mobile authentication with a provider SDK

After everything is configured on the backend, you can modify mobile clients to sign in with App Service. There are two approaches here:

- Use an SDK that a given identity provider publishes to establish identity and then gain access to App Service.
- Use a single line of code so that the Mobile Apps client SDK can sign in users.

TIP

Most applications should use a provider SDK to get a more consistent experience when users sign in, to use refresh support, and to get other benefits that the provider specifies.

When you use a provider SDK, users can sign in to an experience that integrates more tightly with the operating system that the app is running on. This also gives you a provider token and some user information on the client, which makes it much easier to consume graph APIs and customize the user experience. Occasionally on blogs and forums, you will see this referred to as the "client flow" or "client-directed flow" because code on the client signs in users, and the client code has access to a provider token.

After a provider token is obtained, it needs to be sent to App Service for validation. After App Service validates the token, App Service creates a new App Service token that is returned to the client. The Mobile Apps client SDK has helper methods to manage this exchange and automatically attach the token to all requests to the application backend. Developers can also keep a reference to the provider token if they so choose.

Mobile authentication without a provider SDK

If you do not want to set up a provider SDK, you can allow the Mobile Apps feature of Azure App Service to sign in for you. The Mobile Apps client SDK will open a web view to the provider of your choosing and sign in the user. Occasionally on blogs and forums, you will see this referred to as the "server flow" or "server-directed flow" because the server manages the process that signs in users, and the client SDK never receives the provider token.

Code to start this flow is included in the authentication tutorial for each platform. At the end of the flow, the client SDK has an App Service token, and the token is automatically attached to all requests to the application backend.

Service-to-service authentication

Although you can give users access to your application, you can also trust another application to call your own API. For example, you could have one web app call an API in another web app. In this scenario, you use credentials for a service account instead of user credentials to get a token. A service account is also known as a *service principal* in Azure Active Directory parlance, and authentication that uses such an account is also known as a service-to-service scenario.

IMPORTANT

Because mobile apps run on customer devices, mobile applications do *not* count as trusted applications and should not use a service principal flow. Instead, they should use a user flow that was detailed earlier.

For service-to-service scenarios, App Service can protect your application by using Azure Active Directory. The calling application just needs to provide an Azure Active Directory service principal authorization token that is obtained by providing the client ID and client secret from Azure Active Directory. An example of this scenario that uses ASP.NET API apps is explained by the tutorial, [Service principal authentication for API Apps][apia-service].

If you want to use App Service authentication to handle a service-to-service scenario, you can use client certificates or basic authentication. For information about client certificates in Azure, see [How To Configure TLS Mutual Authentication for Web Apps](#). For information about basic authentication in ASP.NET, see [Authentication Filters in ASP.NET Web API 2](#).

Service account authentication from an App Service logic app to an API app is a special case that is detailed in [Using your custom API hosted on App Service with Logic apps](#).

How authorization works in App Service

You have full control over the requests that can access your application. App Service Authentication / Authorization can be configured with any of the following behaviors:

- Allow only authenticated requests to reach your application.

If a browser sends an anonymous request, App Service will redirect to a page for the identity provider that you choose so that users can sign in. If the request comes from a mobile device, the returned response is an **HTTP 401 Unauthorized** response.

With this option, you don't need to write any authentication code at all in your app. If you need finer authorization, information about the user is available to your code.

- Allow all requests to reach your application, but validate authenticated requests, and pass along authentication information in the HTTP headers.

This option defers authorization decisions to your application code. It provides more flexibility in handling anonymous requests, but you have to write code.

- Allow all requests to reach your application, and take no action on authentication information in the requests.

In this case, the Authentication / Authorization feature is off. The tasks of authentication and authorization are entirely up to your application code.

The previous behaviors are controlled by the **Action to take when request is not authenticated** option in the Azure portal. If you choose **Log in with provider name **, all requests have to be authenticated. **Allow request (no action)** defers the authorization decision to your code, but it still provides authentication information. If you want to have your code handle everything, you can disable the Authentication / Authorization feature.

Working with user identities in your application

App Service passes some user information to your application by using special headers. External requests prohibit these headers and will only be present if set by App Service Authentication / Authorization. Some example headers include:

- X-MS-CLIENT-PRINCIPAL-NAME
- X-MS-CLIENT-PRINCIPAL-ID
- X-MS-TOKEN-FACEBOOK-ACCESS-TOKEN
- X-MS-TOKEN-FACEBOOK-EXPIRES-ON

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.

Your application can also obtain additional user details through an HTTP GET on the `/auth/me` endpoint of your application. A valid token that's included with the request will return a JSON payload with details about the provider that's being used, the underlying provider token, and some other user information. The Mobile Apps server SDKs provide helper methods to work with this data. For more information, see [How to use the Azure Mobile Apps Node.js SDK](#), and [Work with the .NET backend server SDK for Azure Mobile Apps](#).

Documentation and additional resources

Identity providers

The following tutorials show how to configure App Service to use different authentication providers:

- [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](#)

If you want to use an identity system other than the ones provided here, you can also use the [preview custom authentication support in the Mobile Apps .NET server SDK](#), which can be used in web apps, mobile apps, or API apps.

Mobile applications

The following tutorials show how to add authentication to your mobile clients by using the server-directed flow:

- [Add authentication to your iOS app](#)
- [Add Authentication to your Android app](#)
- [Add Authentication to your Windows app](#)
- [Add authentication to your Xamarin.iOS app](#)
- [Add authentication to your Xamarin.Android app](#)
- [Add authentication to your Xamarin.Forms app](#)
- [Add Authentication to your Cordova app](#)

Use the following resources if you want to use the client-directed flow for Azure Active Directory:

- [Use the Active Directory Authentication Library for iOS](#)
- [Use the Active Directory Authentication Library for Android](#)
- [Use the Active Directory Authentication Library for Windows and Xamarin](#)

Use the following resources if you want to use the client-directed flow for Facebook:

- [Use the Facebook SDK for iOS](#)

Use the following resources if you want to use the client-directed flow for Twitter:

- [Use Twitter Fabric for iOS](#)

Use the following resources if you want to use the client-directed flow for Google:

- [Use the Google Sign-In SDK for iOS](#)

Azure App Service Hybrid Connections

11/29/2017 • 8 min to read • [Edit Online](#)

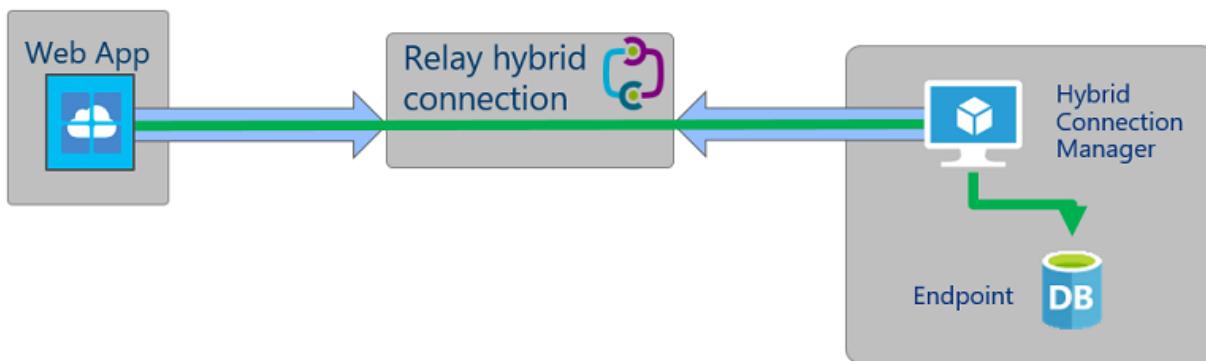
Hybrid Connections is both a service in Azure and a feature in Azure App Service. As a service, it has uses and capabilities beyond those that are used in App Service. To learn more about Hybrid Connections and their usage outside App Service, see [Azure Relay Hybrid Connections](#).

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.

The Hybrid Connections feature has two types: the Hybrid Connections that are offered as a service under Service Bus Relay, and the older Azure BizTalk Services Hybrid Connections. The latter are referred to as Classic Hybrid Connections in the portal. There is more information about them later in this article.

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

There are a few things you cannot do with Hybrid Connections, including:

- Mounting a drive.
- Using UDP.
- Accessing TCP-based services that use dynamic ports, such as FTP Passive Mode or Extended Passive Mode.
- Supporting LDAP, because it sometimes requires UDP.
- Supporting Active Directory.

Add and Create Hybrid Connections in your app

You can create Hybrid Connections through your App Service app in the Azure portal, or from Azure Relay in the Azure portal. We recommend that you create Hybrid Connections through the App Service app that you want to use with the Hybrid Connection. To create a Hybrid Connection, go to the [Azure portal](#) and select your app. Select **Networking > Configure your Hybrid Connection endpoints**. From 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, there's a header with the app name 'hcdemopapp1'. Below the header, there's a 'Refresh' button and a 'Hybrid connections' icon. A descriptive text block says: '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.' Underneath, there are two main sections: 'App service plan (pricing tier)' set to 'hcdemoplan (Standard)', and 'Location' set to 'North Central US'. To the right, there's a summary box showing 'Connections used': 8 active, 17 total, and 0/25 limit. Below these, there's a 'Download connection manager' link and a 'Add hybrid connection' button. The 'Classic hybrid connections' section is labeled 'No results'. At the bottom, there's another 'Add classic hybrid connection' button and a 'Classic hybrid connections' table with columns for NAME, STATUS, HOST, and PORT, also showing 'No results'.

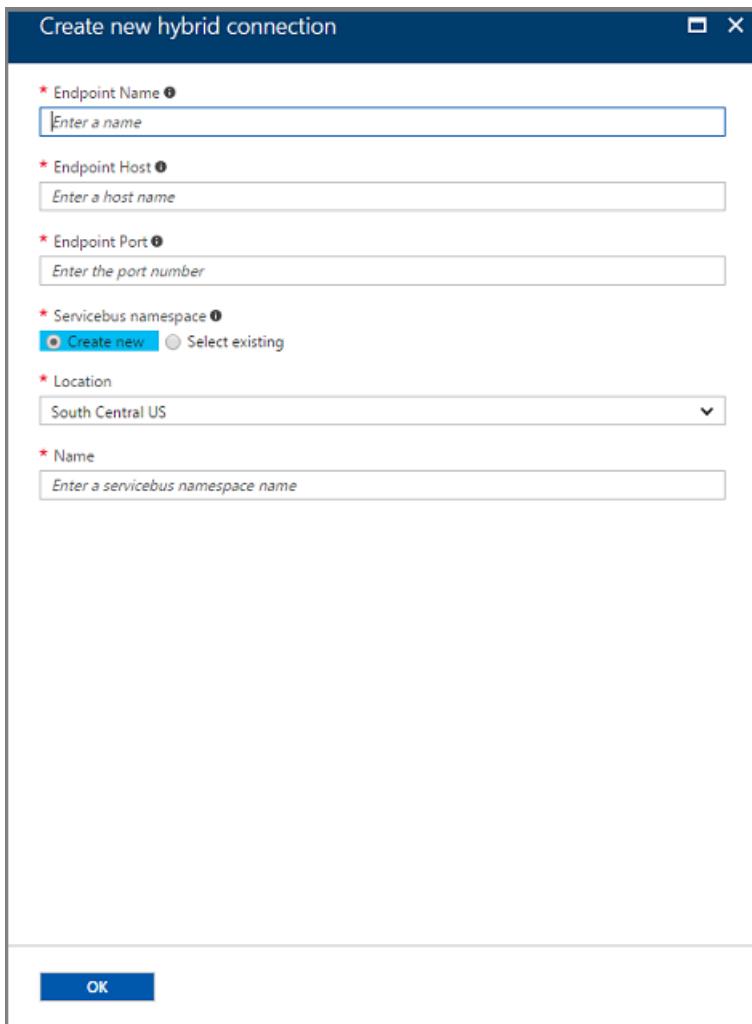
To add a new Hybrid Connection, select **Add hybrid connection**. You'll see a list of the Hybrid Connections that you have already created. To add one or more of them to your app, select the ones you want, and then select **Add selected Hybrid Connection**.

The screenshot shows two windows side-by-side. The left window is titled 'Hybrid connections' and displays general information about the app 'hcdemopl1'. It shows an App Service plan (hcemopl1 Standard) with 8 connections used out of 25 allowed, and a location of North Central US. The right window is titled 'Add hybrid connection' and shows a table of existing hybrid connections. One connection is listed: 'workstation-mysql' with host 'compy-wkstn', port 3306, namespace 'workstation-mysql', and location 'North Central US'.

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:

- Endpoint name.
- Endpoint hostname.
- Endpoint port.
- Service Bus namespace you want to use.



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.

The screenshot shows the Azure Hybrid Connections blade for an app named 'hcdemopl1'. On the left, it displays the 'Hybrid connections' section with a summary of 8 active connections out of 25 allowed. It also shows the app service plan (Standard) and location (North Central US). On the right, the 'Properties' pane provides detailed information about the endpoint: Endpoint Name (workstation-mysql), Endpoint Host (compy-wkstn), and Endpoint Port (3306). The Service Bus Namespace is set to 'workstation-mysql'. The Gateway Connection String is listed as 'Endpoint=sb://purple-north-relay.servicebus.windows.net/'. There are sections for Hybrid Connection Managers (2 connected) and Namespace Location (North Central US).

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

The Hybrid Connections feature is only available in Basic, Standard, Premium, and Isolated pricing SKUs. There are limits tied to the pricing plan.

NOTE

You can only create new Hybrid Connections based on Azure Relay. You cannot create new BizTalk Hybrid Connections.

PRICING PLAN	NUMBER OF HYBRID CONNECTIONS USABLE IN THE PLAN
Basic	5
Standard	25
Premium	200
Isolated	200

Note that the App Service plan shows you how many Hybrid Connections are being used and by what apps.

The screenshot shows two windows side-by-side. The left window is titled 'Hybrid connections' and displays a list of hybrid connections. One connection is selected, showing details like name 'workstation-mysql' and endpoint '3306 : compy-wkstn'. The right window is titled 'Properties' and shows detailed configuration for the selected hybrid connection. It includes fields for Endpoint Name ('workstation-mysql'), Endpoint Host ('compy-wkstn'), Endpoint Port ('3306'), Service Bus Namespace ('workstation-mysql'), Namespace Location ('North Central US'), Connection Manager Connections ('2'), Number of Connected Sites ('1'), and a Gateway Connection String ('Endpoint=sb://purple-north-relay.servicebus.windows.net/').

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.

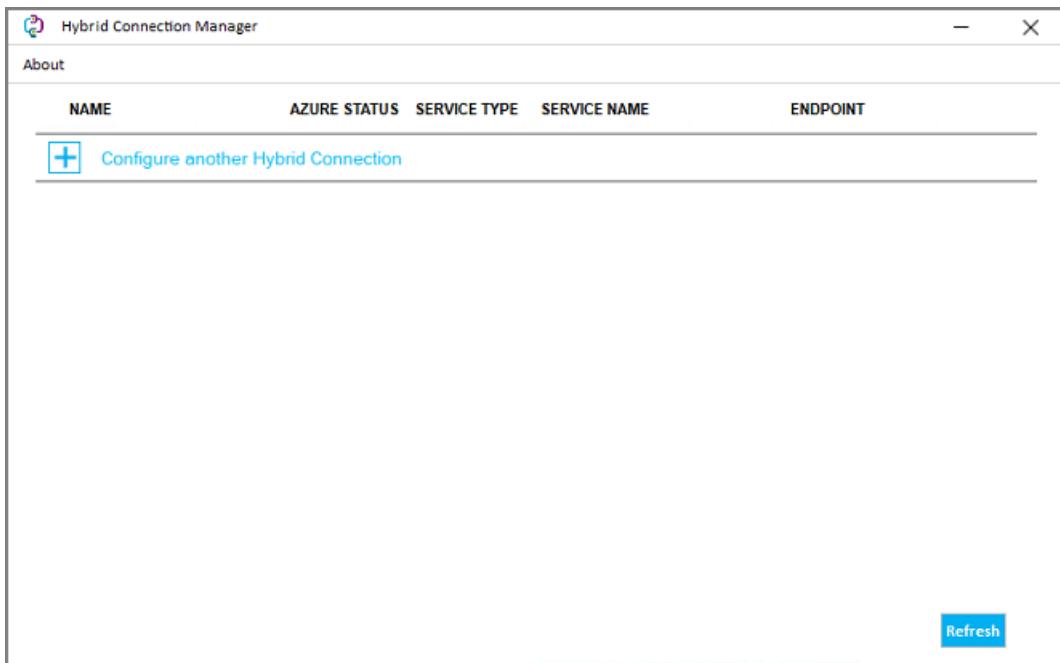
There is an additional cost to using Hybrid Connections. For details, see [Service Bus pricing](#).

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. When installed, HCM runs as a service that connects to Service Bus Relay, based on the configured endpoints. The connections from HCM are outbound to Azure over port 443.

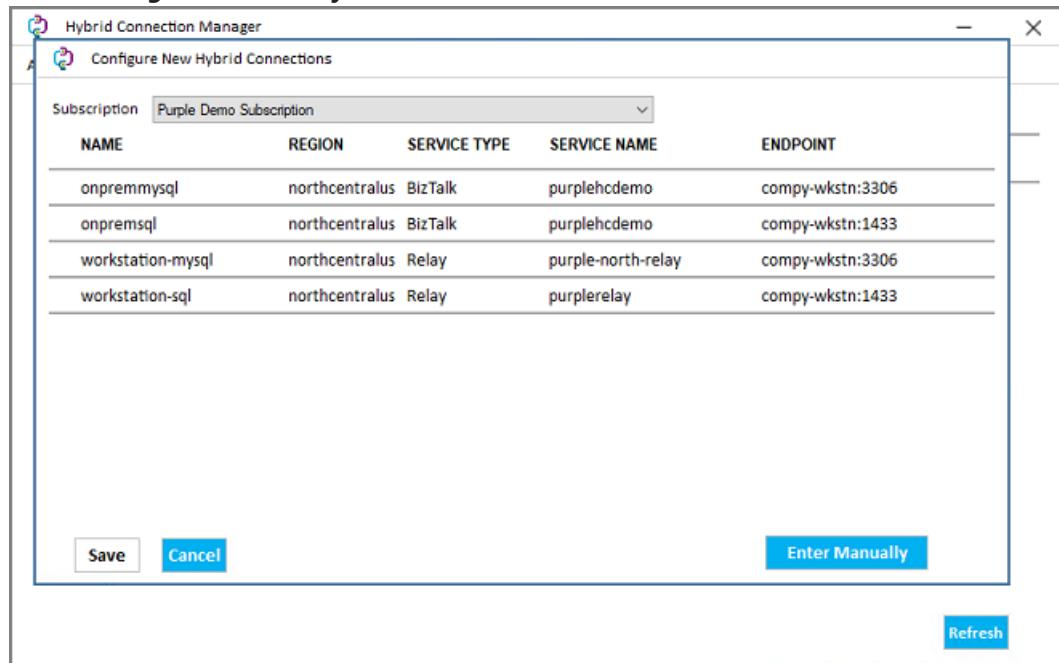
After installing HCM, you can run HybridConnectionManagerUi.exe to use the UI for the tool. This file is in the Hybrid Connection Manager installation directory. In Windows 10, you can also just search for *Hybrid Connection Manager UI* in your search box.



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.
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 header bar with the title 'Hybrid Connection Manager' and standard window controls. Below the header is a section titled 'About'. The main area contains 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'. Above this row, there's a blue button labeled 'Configure another Hybrid Connection'. 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. At the top, it says 'Remove' and has a close button. Below that is 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. Below the dialog is a 'Refresh' button.

To support the Hybrid Connections it is configured with, HCM requires:

- TCP access to Azure over ports 80 and 443.
- TCP access to the Hybrid Connection endpoint.
- The ability to do DNS look-ups on the endpoint host and the Service Bus namespace.

HCM supports both new Hybrid Connections and BizTalk Hybrid Connections.

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.

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 this in the properties for a Hybrid Connection in the [Azure portal](#). To use that string, select **Enter Manually** in the HCM, and paste in the gateway connection string.

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 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.

BizTalk Hybrid Connections

The older BizTalk Hybrid Connections capability has been closed to new BizTalk Hybrid Connections. You can continue using your existing BizTalk Hybrid Connections with your apps, but you should migrate to the new Hybrid Connections that use Azure Relay. Among the benefits in the new service over the BizTalk version are:

- No additional BizTalk account is required.
- TLS is version 1.2 instead of version 1.0.
- Communication is over ports 80 and 443, and uses a DNS name to reach Azure, instead of IP addresses and a range of additional ports.

To add an existing BizTalk Hybrid Connection to your app, go to your app in the [Azure portal](#), and select **Networking > Configure your Hybrid Connection endpoints**. In the Classic Hybrid Connections table, select **Add Classic Hybrid Connection**. You can then see a list of your BizTalk Hybrid Connections.

Controlling Azure App Service traffic with Azure Traffic Manager

11/29/2017 • 3 min 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 three 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 Web Apps](#).
- 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](#).

For more information about using Traffic Manager with App Service, see the blog posts [Using Azure Traffic Manager with Azure Web Sites](#) and [Azure Traffic Manager can now integrate with Azure Web Sites](#).

Azure App Service Local Cache overview

9/26/2017 • 6 min to read • [Edit Online](#)

Azure web app 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 web app.
- The content is durable and can be modified by running web 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 web app (typically some technologies such as ASP.NET do initiate a web app restart on some file changes to get the latest content).

While many web apps use one or all of these features, some web 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. Web 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 Local Cache changes the behavior of App Service

- The local cache is a copy of the /site and /siteextensions folders of the web app. It is created on the local VM instance on web app startup. The size of the local cache per web app is limited to 300 MB by default, but you can increase it up to 2 GB.
- The local cache is read-write. However, any modifications are discarded when the web app moves virtual machines or gets restarted. Do not use Local Cache for apps that store mission-critical data in the content store.
- Web apps can continue to write log files and diagnostic data as they do currently. Log files and data, however, are stored locally on the VM. Then they are copied over periodically to the shared content store. The copy to the shared content store is a best-case effort--write backs could be lost due to a sudden crash of a VM instance.
- There is a change in the folder structure of the LogFiles and Data folders for web apps that use Local Cache. There are now subfolders in the storage LogFiles and Data folders that follow the naming pattern of "unique identifier" + time stamp. Each of the subfolders corresponds to a VM instance where the web app is running or has run.
- Publishing changes to the web app through any of the publishing mechanisms will publish to the shared content store. This is by design because we want the published content to be durable. To refresh the local cache of the web app, it needs to be restarted. Does this seem like an excessive step? To make the lifecycle seamless, see the information later in this article.
- D:\Home points to the local cache. D:\local continues to point to the temporary VM-specific storage.
- 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

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 web 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 web app?

If your web 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 web app does not seem to have them. Why?

If your web 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 web app still gets restarted. Why is that? I thought Local Cache helped with frequent app restarts.

Local Cache does help prevent storage-related web app restarts. However, your web 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 web app.

Azure App Service diagnostics overview

11/15/2017 • 2 min 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 web app with no configuration required. When you do run into issues with your web app, App Service diagnostics will point 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 web app within the last 24 hours, all the diagnostic graphs will be available for you to analyze at all times. Additional troubleshooting tools and links to helpful documentation and forums are located on the right-hand column.

The screenshot shows the Azure App Service Diagnostics interface for a web application named "BuggyBakery". The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems (selected), Deployment (Quickstart, Deployment slots, Deployment options, Continuous Delivery (Preview)), Settings (Application settings, Authentication / Authorization, Managed service identity, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan)), and Support Tools (Metrics per Instance (Apps), Metrics per Instance (App Service Plan), Live HTTP Traffic, Application Events, Failed Request Tracing Logs, Diagnostics as a Service, Mitigate, Advanced Application Restart). The main content area is titled "Home" and features four status cards: "Web App Down" (blue), "Web App Slow" (red), "High CPU Usage" (purple), and "High Memory Usage" (green). Below these cards is a callout box asking if a health checkup should be performed. A detailed description of the health checkup process follows. At the bottom is a line chart titled "Requests and Errors" showing request count over time, with tabs for Requests and Errors, App Performance, CPU Usage, and Memory Usage. The chart includes a legend for Requests (blue), Http 2xx (green), Http 3xx (purple), Http 4xx (orange), and Http Server Errors (red).

Health checkup

If you don't know what's wrong with your web app or don't know where to start troubleshooting your issues, the health checkup is a good place to start. The health checkup will analyze your web 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.

Home

Once your health checkup is complete, please use the tabs to navigate between the different categories. Click 'View Full Report' to get more details and potential quick solutions and troubleshooting advice.

Requests and Errors

App Performance

CPU Usage

Memory Usage

Request Count

Time (UTC)

Your Web App is currently experiencing HTTP server errors. Please "View Full Report" to see more detailed observations and quick solutions.

[View Full Report >](#)

I noticed that your app was experiencing high CPU usage within the last 24 hours. Would you like me to show you more details about the issues we found?

SUPPORT TOOLS

- Metrics per Instance (Apps)
- Metrics per Instance (App Service Plan)
- Live HTTP Traffic
- Application Events
- Failed Request Tracing Logs
- Diagnostics as a Service
- Mitigate
- Advanced Application Restart

PREMIUM TOOLS

FAQs

- Application Performance FAQs
- Deployment FAQs
- OSS FAQs
- Configuration and Management FAQs

RESOURCE CENTER

COMMUNITY

- MSDN Forums
- Stack Overflow
- @AzureSupport

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.

Home

Would you like to see the troubleshooting suggestions that I have tailored to your specific issue?

Troubleshooting and Next Steps

Restart App
Mitigation

Scale Up App Service Plan
Mitigation

Yes!

Perform a Web App Restart

Mitigation

If your app is in a bad state, performing a web app restart can be enough to fix the problem in some cases. There are different types of restart, Advanced Application Restart and the standard App Restart

Option #1: Advanced Application Restart

Advanced Application Restart is the perfect solution if your app is running on multiple instances but you only want to restart your app on specific instances.

We detected the following site as the App to be Restarted: **buggybakery**
Unhealthy Instance(s): RD00155D3BEAA6

[Restart App On Specified Instances](#)

Option #2: App Restart

An App Restart will kill the app process on all instances.

SUPPORT TOOLS

- Metrics per Instance (Apps)
- Metrics per Instance (App Service Plan)
- Live HTTP Traffic
- Application Events
- Failed Request Tracing Logs
- Diagnostics as a Service
- Mitigate
- Advanced Application Restart

PREMIUM TOOLS

FAQs

- Application Performance FAQs
- Deployment FAQs
- OSS FAQs
- Configuration and Management FAQs

RESOURCE CENTER

COMMUNITY

- MSDN Forums
- Stack Overflow
- @AzureSupport

Tile shortcuts

If you know exactly what kind of troubleshooting information you're looking for, the tile shortcuts will take you directly to the full diagnostic report of the problem category that you're interested in. Compared to the health checkup, the tile shortcuts are the more direct, but less guided way of accessing your diagnostic metrics.

The screenshot shows the 'Home' tab selected in the top navigation bar. A message box says, 'Hello! Welcome to App Service diagnostics! I'm here to help you diagnose and solve problems with your Web App.' Below it, another message box says, 'Here are some of the things that I can help you with:' followed by five colored buttons labeled: 'Web App Down' (blue), 'Web App Slow' (pink), 'High CPU Usage' (purple), 'High Memory Usage' (green), and 'Web App Restarted' (orange). A red box highlights the 'Web App Down' button. A third message box asks, 'First, would you like me to perform a health checkup on your Web App?' with two buttons below: 'Yes' (blue) and 'Maybe Later' (orange).

Diagnostic report

Whether you want more information after running a [health checkup](#) or you clicked on one of the [tile shortcuts](#), the full diagnostic report will show you relevant graphed metrics from the last 24 hours. If your app experiences any downtime, it's represented by an orange bar underneath the timeline. You can select one of the downtimes to get analyzed observations about the downtime and the suggested solutions.

The screenshot shows the 'App Error Analysis' tab selected in the top navigation bar. It displays a timeline chart for the last 24 hours. The Y-axis is 'Percent' (0.00 to 100.00) and the X-axis is 'Time (UTC)'. A blue line represents 'Platform Availability' and a red line represents 'App Availability'. Orange horizontal bars at the bottom indicate periods of downtime. A location pin icon is at the end of the last bar. At the top, there are status indicators: 'Your app is experiencing server errors' (red exclamation mark) and 'Platform is healthy' (green checkmark). To the right, it says 'Last updated: 11/10/2017, 7:49:07 AM (UTC)' and has a 'Refresh' button. Below the chart, a message says 'Your app is unhealthy right now' and 'High CPU and 3 other events detected.' with a 'Less Details' link. A 'Observations:' section lists several bullet points with 'View Details' links. At the bottom, a table shows 'Uri', 'Percentage', and 'Count' for specific events.

Open App Service diagnostics

To access App Service diagnostics, navigate to your App Service web app in the [Azure portal](#).

In the left navigation, click on **Diagnose and solve problems**.

Configure web apps in Azure App Service

1/2/2018 • 6 min to read • [Edit Online](#)

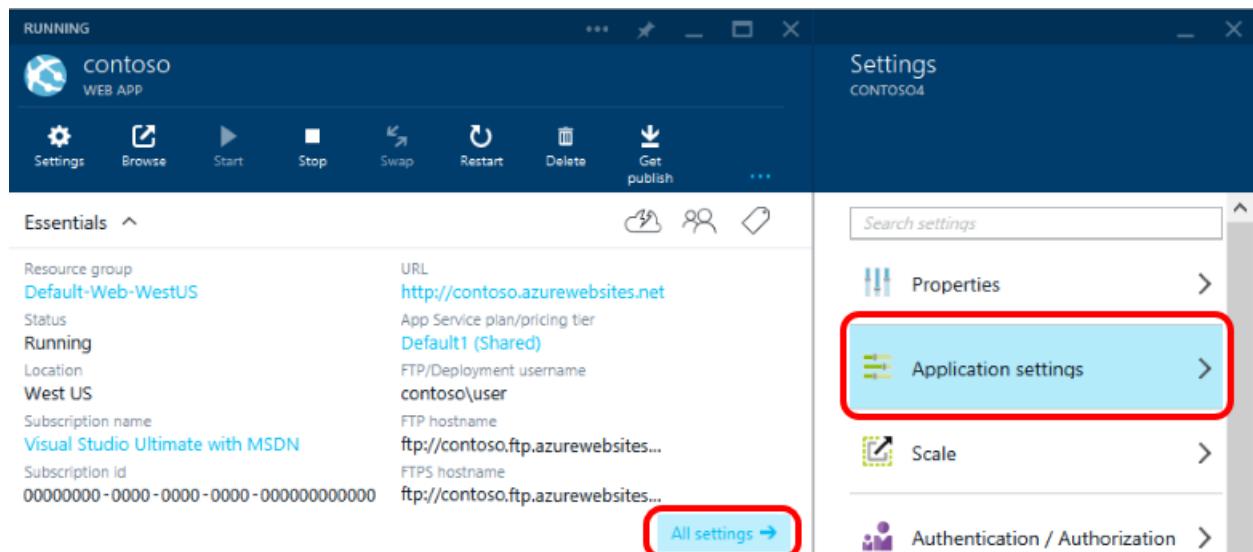
This topic explains how to configure a web app using the [Azure Portal](#).

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

Application settings

1. In the [Azure Portal](#), open the blade for the web app.
2. Click **Application settings**.



The **Application settings** blade has settings grouped under several categories.

General settings

Framework versions. Set these options if your app uses any these frameworks:

- **.NET Framework:** Set the .NET framework version.
- **PHP:** Set the PHP version, or **OFF** to disable PHP.
- **Java:** Select the Java version or **OFF** to disable Java. Use the **Web Container** option to choose between Tomcat and Jetty versions.
- **Python:** Select the Python version, or **OFF** to disable Python.

For technical reasons, enabling Java for your app disables the .NET, PHP, and Python options.

Platform. Selects whether your web app runs in a 32-bit or 64-bit environment. The 64-bit environment requires Basic or Standard mode. Free and Shared modes always run in a 32-bit environment.

NOTE

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

Web Sockets. Set **ON** to enable the WebSocket protocol; for example, if your web app uses [ASP.NET SignalR](#) or [socket.io](#).

Always On. By default, web apps are unloaded if they are idle for some period of time. This lets the system conserve resources. In Basic or Standard mode, you can enable **Always On** to keep the app loaded all the time. If your app runs continuous WebJobs or runs WebJobs triggered using a CRON expression, you should enable **Always On**, or the web jobs may not run reliably.

Managed Pipeline Version. Sets the IIS [pipeline mode](#). Leave this set to Integrated (the default) unless you have a legacy app that requires an older version of IIS.

Auto Swap. If you enable Auto Swap for a deployment slot, App Service will automatically swap the web app into production when you push an update to that slot. For more information, see [Deploy to staging slots for web apps in Azure App Service](#).

Debugging

Remote Debugging. Enables remote debugging. When enabled, you can use the remote debugger in Visual Studio to connect directly to your web app. Remote debugging will remain enabled for 48 hours.

App settings

This section contains name/value pairs that your web app will load on start up.

- For .NET apps, these settings are injected into your .NET configuration `AppSettings` at runtime, overriding existing settings.
- PHP, Python, Java and Node applications can access these settings as environment variables at runtime. For each app setting, two environment variables are created; one with the name specified by the app setting entry, and another with a prefix of `APPSETTING_`. Both contain the same value.

Connection strings

Connection strings for linked resources.

For .NET apps, these connection strings are injected into your .NET configuration `connectionStrings` settings at runtime, overriding existing entries where the key equals the linked database name.

For PHP, Python, Java and Node applications, these settings will be available as environment variables at runtime, prefixed with the connection type. The environment variable prefixes are as follows:

- SQL Server: `SQLCONNSTR_`
- MySQL: `MYSQLCONNSTR_`
- SQL Database: `SQLAZURECONNSTR_`
- Custom: `CUSTOMCONNSTR_`

For example, if a MySQL connection string were named `connectionstring1`, it would be accessed through the environment variable `MYSQLCONNSTR_connectionString1`.

Default documents

The default document is the web page that is displayed at the root URL for a website. The first matching file in the list is used.

Web apps might use modules that route based on URL, rather than serving static content, in which case there is no default document as such.

Handler mappings

Use this area to add custom script processors to handle requests for specific file extensions.

- Extension.** The file extension to be handled, such as `*.php` or `handler.fcgi`.
- Script Processor Path.** The absolute path of the script processor. Requests to files that match the file extension

will be processed by the script processor. Use the path `D:\home\site\wwwroot` to refer to your app's root directory.

- **Additional Arguments.** Optional command-line arguments for the script processor

Virtual applications and directories

To configure virtual applications and directories, specify each virtual directory and its corresponding physical path relative to the website root. Optionally, you can select the **Application** checkbox to mark a virtual directory as an application.

Enabling diagnostic logs

To enable diagnostic logs:

1. In the blade for your web app, click **All settings**.
2. Click **Diagnostic logs**.

Options for writing diagnostic logs from a web application that supports logging:

- **Application Logging.** Writes application logs to the file system. Logging lasts for a period of 12 hours.

Level. When application logging is enabled, this option specifies the amount of information that will be recorded (Error, Warning, Information, or Verbose).

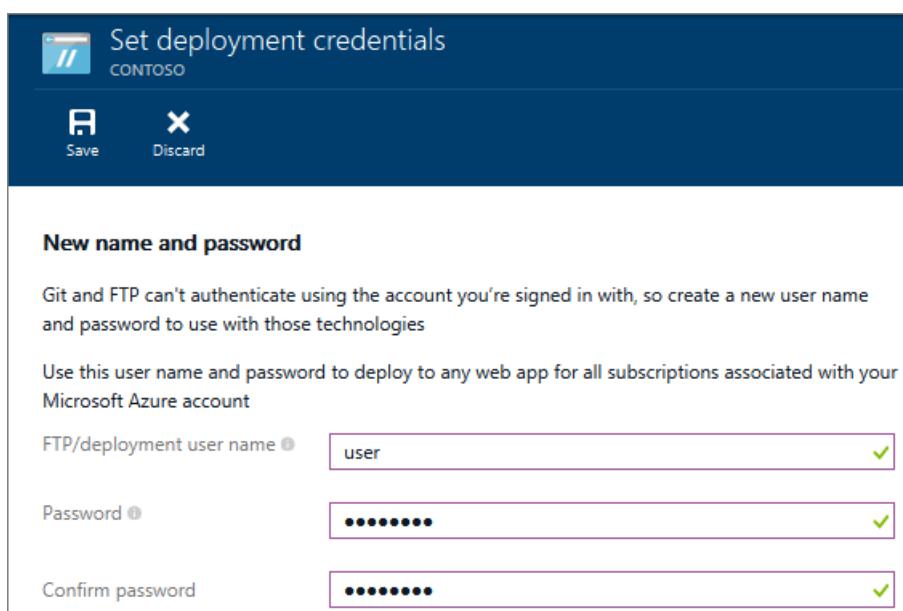
Web server logging. Logs are saved in the W3C extended log file format.

Detailed error messages. Saves detailed error messages .htm files. The files are saved under /LogFiles/DetailedErrors.

Failed request tracing. Logs failed requests to XML files. The files are saved under /LogFiles/W3SVCxxx, where xxx is a unique identifier. This folder contains an XSL file and one or more XML files. Make sure to download the XSL file, because it provides functionality for formatting and filtering the contents of the XML files.

To view the log files, you must create FTP credentials, as follows:

1. In the blade for your web app, click **All settings**.
2. Click **Deployment credentials**.
3. Enter a user name and password.
4. Click **Save**.



The full FTP user name is "app\username" where *app* is the name of your web app. The username is listed in the

web app blade, under **Essentials**.

The screenshot shows the 'Essentials' section of the Azure Web App blade. It displays various configuration details:

Setting	Value
Resource group	Default-Web-WestUS
Status	Running
Location	West US
Subscription name	Visual Studio Ultimate with MSDN
Subscription id	00000000-0000-0000-0000-000000000000
URL	http://contoso.azurewebsites.net
App Service plan/pricing tier	Default1 (Shared)
FTP/Deployment username	contoso\user
FTP hostname	ftp://contoso.ftp.azurewebsites...
FTPS hostname	ftps://contoso.ftp.azurewebsites...

[All settings →](#)

Other configuration tasks

SSL

In Basic or Standard mode, you can upload SSL certificates for a custom domain. For more information, see [Enable HTTPS for a web app].

To view your uploaded certificates, click **All Settings > Custom domains and SSL**.

Domain names

Add custom domain names for your web app. For more information, see [Configure a custom domain name for a web app in Azure App Service].

To view your domain names, click **All Settings > Custom domains and SSL**.

Deployments

- Set up continuous deployment. See [Using Git to deploy Web Apps in Azure App Service](#).
- Deployment slots. See [Deploy to Staging Environments for Web Apps in Azure App Service](#).

To view your deployment slots, click **All Settings > Deployment slots**.

Monitoring

In Basic or Standard mode, you can test the availability of HTTP or HTTPS endpoints, from up to three geo-distributed locations. A monitoring test fails if the HTTP response code is an error (4xx or 5xx) or the response takes more than 30 seconds. An endpoint is considered available if the monitoring tests succeed from all the specified locations.

For more information, see [How to: Monitor web endpoint status](#).

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Next steps

- [Configure a custom domain name in Azure App Service](#)
- [Enable HTTPS for an app in Azure App Service](#)
- [Scale a web app in Azure App Service](#)
- [Monitoring basics for Web Apps in Azure App Service](#)

Add a Java application to Azure App Service Web Apps

9/19/2017 • 1 min to read • [Edit Online](#)

Once you have initialized your Java web app in [Azure App Service](#) as documented at [Create a Java web app in Azure App Service](#), you can upload your application by placing your WAR in the **webapps** folder.

The navigation path to the **webapps** folder differs based on how you set up your Web Apps instance.

- If you set up your web app by using the Azure Marketplace, the path to the **webapps** folder is in the form **d:\home\site\wwwroot\bin\application_server\webapps**, where **application_server** is the name of the application server in effect for your Web Apps instance.
- If you set up your web app by using the Azure configuration UI, the path to the **webapps** folder is in the form **d:\home\site\wwwroot\webapps**.

Note that you can use source control to upload your application or web pages, including [continuous integration scenarios](#). FTP is also an option for uploading your application or web pages; for more information about deploying your applications over FTP, see [Deploy your app using FTP](#).

Note for Tomcat web apps: Once you've uploaded your WAR file to the **webapps** folder, the Tomcat application server will detect that you've added it and will automatically load it. Note that if you copy files (other than WAR files) to the ROOT directory, the application server will need to be restarted before those files are used. The autoload functionality for the Tomcat Java web apps running on Azure is based on a new WAR file being added, or new files or directories added to the **webapps** folder.

See Also

For more information about using Azure with Java, see the [Azure Java Developer Center](#).

[application-insights-app-insights-java-get-started](#)

Configure PHP in Azure App Service Web Apps

9/19/2017 • 6 min to read • [Edit Online](#)

Introduction

This guide will show you how to configure the built-in PHP runtime for Web 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 web app in App Service.

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

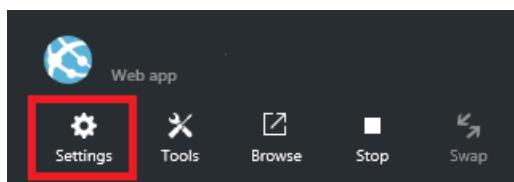
How to: Change the built-in PHP version

By default, PHP 5.5 is installed and immediately available for use when you create an App Service web 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 5.6 and PHP 7.0 versions are also available, but not enabled by default. To update the PHP version, follow one of these methods:

Azure Portal

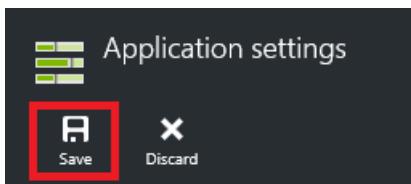
1. Browse to your web app in the [Azure Portal](#) and click on the **Settings** button.



2. From the **Settings** blade select **Application Settings** and choose the new PHP version.

The screenshot shows the 'Application settings' blade in the Azure portal. At the top, there are 'Save' and 'Discard' buttons. Below them is a section titled 'General settings' with a note about enabling 64-bit and Always On options. The 'PHP version' setting is highlighted with a red box, showing '5.4'. Other settings include '.NET Framework version' (v4.6), 'Java version' (Off), 'Python version' (Off), and 'Platform' (32-bit).

3. Click the **Save** button at the top of the **Web app settings** blade.



Azure PowerShell (Windows)

1. Open Azure PowerShell, and login to your account:

```
PS C:\> Login-AzureRmAccount
```

2. Set the PHP version for the web app.

```
PS C:\> Set-AzureWebsite -PhpVersion {5.5 | 5.6 | 7.0} -Name {app-name}
```

3. The PHP version is now set. You can confirm these settings:

```
PS C:\> Get-AzureWebsite -Name {app-name} | findstr PhpVersion
```

Azure Command-Line Interface (Linux, Mac, Windows)

To use the Azure Command-Line Interface, you must have **Node.js** installed on your computer.

1. Open Terminal, and login to your account.

```
azure login
```

2. Set the PHP version for the web app.

```
azure site set --php-version {5.5 | 5.6 | 7.0} {app-name}
```

- The PHP version is now set. You can confirm these settings:

```
azure site show {app-name}
```

NOTE

The [Azure CLI 2.0](#) commands that are equivalent to the above are:

```
az login
az appservice web config update --php-version {5.5 | 5.6 | 7.0} -g {resource-group-name} -n {app-name}
az appservice web config show -g {resource-group-name} -n {app-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 the steps below. (For information about php.ini directives, see [List of php.ini directives](#).)

Changing PHP_INI_USER, PHP_INI_PERDIR, PHP_INI_ALL configuration settings

- Add a `.user.ini` file to your root directory.
- 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 the `display_errors` setting on 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
```

- Deploy your web app.
- Restart the web 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 web 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

- Add an App Setting to your Web App with the key `PHP_INI_SCAN_DIR` and value `d:\home\site\ini`
- Create an `settings.ini` file using Kudu Console (<http://<site-name>.scm.azurewebsite.net>) in the `d:\home\site\ini` directory.
- 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
```

- Restart your Web App to load the changes.

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, follow the steps below:

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 Web 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.

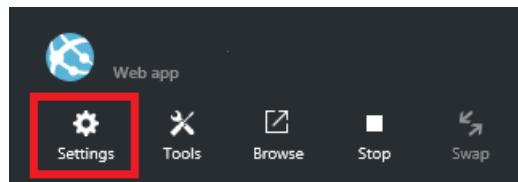
For example, if you wanted to enable the MongoDB and XDebug extensions, your `extensions.ini` file would contain this text:

```
; Enable Extensions  
extension=d:\home\site\ext\php_mongo.dll  
zend_extension=d:\home\site\ext\php_xdebug.dll
```

6. Restart your Web App to load the changes.

Configure via App Setting

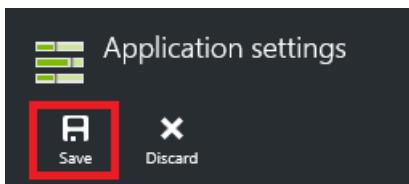
1. Add a `bin` directory to the root directory.
2. Put `.dll` extension files in the `bin` directory (for example, `php_xdebug.dll`). Make sure that the extensions are compatible with default version of PHP and are VC9 and non-thread-safe (nts) compatible.
3. Deploy your web app.
4. Browse to your web app in the Azure Portal and click on the **Settings** button.



5. From the **Settings** blade select **Application Settings** and scroll to the **App settings** section.
6. In the **App settings** section, create a **PHP_EXTENSIONS** key. The value for this key would be a path relative to website root: `bin\your-ext-file`.

The screenshot shows the Azure Portal's 'Settings' blade. On the left, there's a sidebar with links like 'Check health', 'Troubleshoot', 'New support request', 'Quick start', 'Properties', and 'Application settings'. The 'Application settings' link is highlighted with a red box. On the right, the 'Application settings' blade is open, showing sections for 'Auto Swap Slot', 'Debugging' (with 'Remote debugging' set to 'Off'), 'Remote Visual Studio version' (set to 2012), and 'App settings'. Under 'App settings', there's a table with a row for 'WEBSITE_NODE_DEFAULT_V...' (value '4.2.3'). This row has a red box around it, specifically highlighting the 'Key' column ('PHP_EXTENSIONS') and the 'Value' column ('bin\php_mon'). Below this table is another section for 'Connection strings'.

7. Click the **Save** button at the top of the **Web app settings** blade.

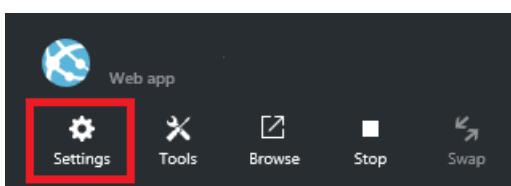


Zend extensions are also supported by using a **PHP_ZENDEXTENSIONS** key. To enable multiple extensions, include a comma-separated list of `.dll` files for the app setting value.

How to: Use a custom PHP runtime

Instead of the default PHP runtime, App Service Web Apps 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 Web Apps, follow the steps below.

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: <http://windows.php.net/download/>. Older releases can be found in the archive here: <http://windows.php.net/downloads/releases/archives/>.
2. Modify the `php.ini` file for your runtime. Note that any configuration settings that are system-level-only directives will be ignored by Web Apps. (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 web app.
6. Browse to your web app in the Azure Portal and click on the **Settings** button.



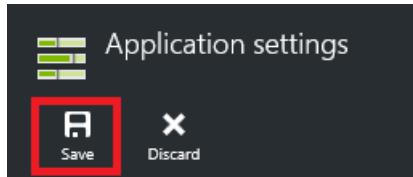
7. From the **Settings** blade select **Application Settings** and scroll to the **Handler mappings** section. Add `*.php` to the Extension field and add the path to the `php-cgi.exe` executable. If you put your PHP runtime in the `bin` directory in the root of your application, the path will be `D:\home\site\wwwroot\bin\php\php-cgi.exe`.

Handler mappings

*.php	D:\home\site\wwwroot\bin\p...	
-------	-------------------------------	--

Extension Processor path Additional arguments

8. Click the **Save** button at the top of the **Web app settings** blade.



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.

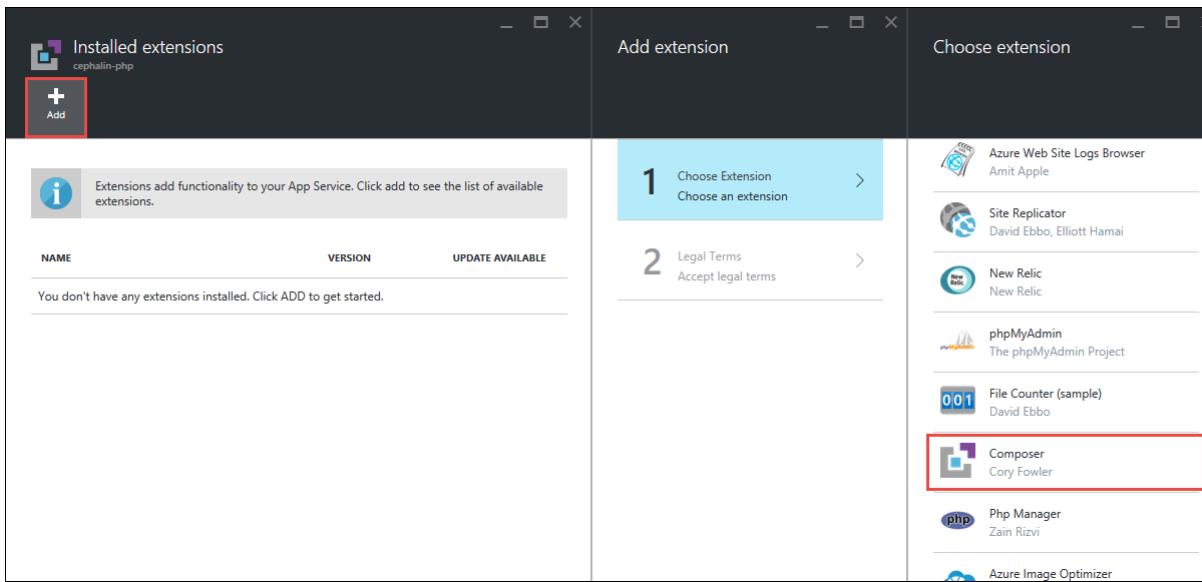
NOTE

You can [vote for first-class Composer support in App Service here!](#)

1. In your PHP web app's blade in the [Azure portal](#), click **Tools > Extensions**.

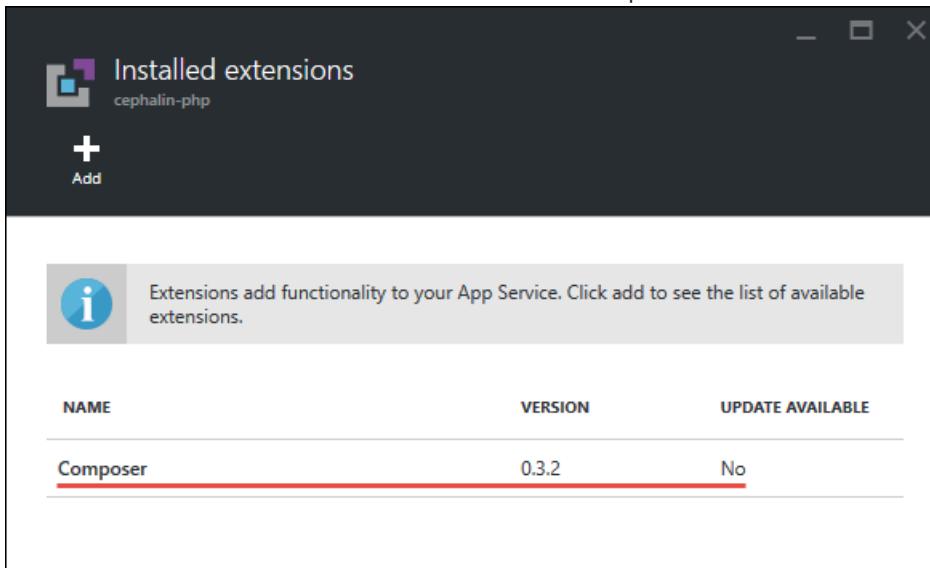
The screenshot shows the Azure portal interface. On the left, there is a blade for a "cephalin-php" web app. The "Tools" tab is selected and highlighted with a red box. On the right, a separate "Tools" blade is open, displaying a list of extensions. The "Extensions" item in this list is also highlighted with a red box.

2. Click **Add**, then click **Composer**.



- Click **OK** to accept legal terms. Click **OK** again to add the extension.

The **Installed extensions** blade will now show the Composer extension.



- Now, perform `git add`, `git commit`, and `git push` like in the previous section. You'll now see 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](#).

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Configuring Python with Azure App Service Web Apps

11/10/2017 • 12 min to read • [Edit Online](#)

This tutorial describes options for authoring and configuring a basic Web Server Gateway Interface (WSGI) compliant Python application on [Azure App Service Web Apps](#).

It describes additional features of Git deployment, such as virtual environment and package installation using requirements.txt.

Bottle, Django, or Flask?

The Azure Marketplace contains templates for the Bottle, Django, and Flask frameworks. If you are developing your first web app in Azure App Service, you can create one quickly from the Azure portal:

- [Creating web apps with Bottle](#)
- [Creating web apps with Django](#)
- [Creating web apps with Flask](#)

Web app creation on Azure portal

This tutorial assumes an existing Azure subscription and access to the Azure portal.

If you do not have an existing web app, you can create one from the [Azure portal](#). Click the NEW button in the top left corner, then click **Web + Mobile** > **Web app**.

Git Publishing

Configure Git publishing for your newly created web app by following the instructions at [Local Git Deployment to Azure App Service](#). This tutorial uses Git to create, manage, and publish your Python web app to Azure App Service.

Once Git publishing is set up, a Git repository is created and associated with your web app. The repository's URL is displayed and can be used to push data from the local development environment to the cloud. To publish applications via Git, make sure a Git client is also installed and use the instructions provided to push your web app content to Azure App Service.

Application Overview

In the next sections, the following files are created. They should be placed in the root of the Git repository.

```
app.py  
requirements.txt  
runtime.txt  
web.config  
ptvs_virtualenv_proxy.py
```

WSGI Handler

WSGI is a Python standard described by [PEP 3333](#) defining an interface between the web server and Python. It provides a standardized interface for writing various web applications and frameworks using Python. Popular

Python web frameworks today use WSGI. Azure App Service Web Apps gives you support for any such frameworks; in addition, advanced users can even author their own as long as the custom handler follows the WSGI specification guidelines.

Here's an example of an `app.py` that defines a custom handler:

```
def wsgi_app(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    response_body = 'Hello World'
    yield response_body.encode()

if __name__ == '__main__':
    from wsgiref.simple_server import make_server

    httpd = make_server('localhost', 5555, wsgi_app)
    httpd.serve_forever()
```

You can run this application locally with `python app.py`, then browse to `http://localhost:5555` in your web browser.

Virtual Environment

Although the preceding example app doesn't require any external packages, it is likely that your application requires some.

To help manage external package dependencies, Azure Git deployment supports the creation of virtual environments.

When Azure detects a `requirements.txt` in the root of the repository, it automatically creates a virtual environment named `env`. This only occurs on the first deployment, or during any deployment after the selected Python runtime has changed.

You probably want to create a virtual environment locally for development, but don't include it in your Git repository.

Package Management

Packages listed in `requirements.txt` are installed automatically in the virtual environment using pip. This happens on every deployment, but pip skips installation if a package is already installed.

Example `requirements.txt`:

```
azure==0.8.4
```

Python Version

Azure will determine the version of Python to use for its virtual environment with the following priority:

1. version specified in `runtime.txt` in the root folder
2. version specified by Python setting in the web app configuration (the **Settings > Application Settings** blade for your web app in the Azure Portal)
3. python-2.7 is the default if none of the above are specified

Valid values for the contents of

```
\runtime.txt
```

are:

- python-2.7
- python-3.4

If the micro version (third digit) is specified, it is ignored.

Example `runtime.txt` :

```
python-2.7
```

Web.config

You need to create a `web.config` file to specify how the server should handle requests.

If you have a `web.x.y.config` file in your repository, where `x.y` matches the selected Python runtime, then Azure automatically copies the appropriate file as `web.config`.

The following `web.config` examples rely on a virtual environment proxy script, which is described in the next section. They work with the WSGI handler used in the example `app.py` above.

Example `web.config` for Python 2.7:

```

<?xml version="1.0"?>
<configuration>
    <appSettings>
        <add key="WSGI_ALT_VIRTUALENV_HANDLER" value="app.wsgi_app" />
        <add key="WSGI_ALT_VIRTUALENV_ACTIVATE_THIS"
            value="D:\home\site\wwwroot\env\Scripts\activate_this.py" />
        <add key="WSGI_HANDLER"
            value="ptvs_virtualenv_proxy.get_virtualenv_handler()" />
        <add key="PYTHONPATH" value="D:\home\site\wwwroot" />
    </appSettings>
    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
    <system.webServer>
        <modules runAllManagedModulesForAllRequests="true" />
        <handlers>
            <remove name="Python27_via_FastCGI" />
            <remove name="Python34_via_FastCGI" />
            <add name="Python FastCGI"
                path="handler.fcgi"
                verb="*"
                modules="FastCgiModule"
                scriptProcessor="D:\Python27\python.exe|D:\Python27\Scripts\wfastcgi.py"
                resourceType="Unspecified"
                requireAccess="Script" />
        </handlers>
        <rewrite>
            <rules>
                <rule name="Static Files" stopProcessing="true">
                    <conditions>
                        <add input="true" pattern="false" />
                    </conditions>
                </rule>
                <rule name="Configure Python" stopProcessing="true">
                    <match url="(.*)" ignoreCase="false" />
                    <conditions>
                        <add input="{REQUEST_URI}" pattern="^/static/.*" ignoreCase="true" negate="true" />
                    </conditions>
                    <action type="Rewrite"
                        url="handler.fcgi/{R:1}"
                        appendQueryString="true" />
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>

```

Example `web.config` for Python 3.4:

```

<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="WSGI_ALT_VIRTUALENV_HANDLER" value="app.wsgi_app" />
    <add key="WSGI_ALT_VIRTUALENV_ACTIVATE_THIS"
         value="D:\home\site\wwwroot\env\Scripts\python.exe" />
    <add key="WSGI_HANDLER"
         value="ptvs_virtualenv_proxy.get_venv_handler()" />
    <add key="PYTHONPATH" value="D:\home\site\wwwroot" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
    <handlers>
      <remove name="Python27_via_FastCGI" />
      <remove name="Python34_via_FastCGI" />
      <add name="Python FastCGI"
           path="handler.fcgi"
           verb="*"
           modules="FastCgiModule"
           scriptProcessor="D:\Python34\python.exe|D:\Python34\Scripts\wfastcgi.py"
           resourceType="Unspecified"
           requireAccess="Script" />
    </handlers>
    <rewrite>
      <rules>
        <rule name="Static Files" stopProcessing="true">
          <conditions>
            <add input="true" pattern="false" />
          </conditions>
        </rule>
        <rule name="Configure Python" stopProcessing="true">
          <match url="(.*)" ignoreCase="false" />
          <conditions>
            <add input="{REQUEST_URI}" pattern="^/static/.*" ignoreCase="true" negate="true" />
          </conditions>
          <action type="Rewrite" url="handler.fcgi/{R:1}" appendQueryString="true" />
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
</configuration>

```

Static files are handled by the web server directly, without going through Python code, for improved performance.

In the preceding examples, the location of the static files on disk should match the location in the URL. This means that a request for `http://pythonapp.azurewebsites.net/static/site.css` will serve the file on disk at `\static\site.css`.

`WSGI_ALT_VIRTUALENV_HANDLER` is where you specify the WSGI handler. In the preceding examples, it's `app.wsgi_app` because the handler is a function named `wsgi_app` in `app.py` in the root folder.

`PYTHONPATH` can be customized, but if you install all your dependencies in the virtual environment by specifying them in `requirements.txt`, you shouldn't need to change it.

Virtual Environment Proxy

The following script is used to retrieve the WSGI handler, activate the virtual environment and log errors. It is designed to be generic and used without modifications.

Contents of `ptvs_virtualenv_proxy.py`:

```
# #####Copyright (c) Microsoft Corporation.
#
# This source code is subject to terms and conditions of the Apache License, Version 2.0. A
# copy of the license can be found in the License.html file at the root of this distribution. If
# you cannot locate the Apache License, Version 2.0, please send an email to
# vspython@microsoft.com. By using this source code in any fashion, you are agreeing to be bound
# by the terms of the Apache License, Version 2.0.
#
# You must not remove this notice, or any other, from this software.
#
# #####
import datetime
import os
import sys
import traceback

if sys.version_info[0] == 3:
    def to_str(value):
        return value.decode(sys.getfilesystemencoding())

    def execfile(path, global_dict):
        """Execute a file"""
        with open(path, 'r') as f:
            code = f.read()
        code = code.replace('\r\n', '\n') + '\n'
        exec(code, global_dict)
else:
    def to_str(value):
        return value.encode(sys.getfilesystemencoding())

def log(txt):
    """Logs fatal errors to a log file if WSGI_LOG env var is defined"""
    log_file = os.environ.get('WSGI_LOG')
    if log_file:
        f = open(log_file, 'a+')
        try:
            f.write('%s: %s' % (datetime.datetime.now(), txt))
        finally:
            f.close()

ptvsd_secret = os.getenv('WSGI_PTVSD_SECRET')
if ptvsd_secret:
    log('Enabling ptvsd ...\\n')
    try:
        import ptvsd
        try:
            ptvsd.enable_attach(ptvsd_secret)
            log('ptvsd enabled.\\n')
        except:
            log('ptvsd.enable_attach failed\\n')
    except ImportError:
        log('error importing ptvsd.\\n')

def get_wsgi_handler(handler_name):
    if not handler_name:
        raise Exception('WSGI_ALT_VIRTUALENV_HANDLER env var must be set')

    if not isinstance(handler_name, str):
        handler_name = to_str(handler_name)

    module_name, _, callable_name = handler_name.rpartition('.')
    should_call = callable_name.endswith('()')
    callable_name = callable_name[:-2] if should_call else callable_name
    name_list = [(callable_name, should_call)]
    handler = None
```

```

last_tb = ''

while module_name:
    try:
        handler = __import__(module_name, fromlist=[name_list[0][0]])
        last_tb = ''
        for name, should_call in name_list:
            handler = getattr(handler, name)
            if should_call:
                handler = handler()
        break
    except ImportError:
        module_name, _, callable_name = module_name.rpartition('.')
        should_call = callable_name.endswith('()')
        callable_name = callable_name[:-2] if should_call else callable_name
        name_list.insert(0, (callable_name, should_call))
        handler = None
        last_tb = ':' + traceback.format_exc()

if handler is None:
    raise ValueError('"%s" could not be imported% % (handler_name, last_tb)')

return handler

activate_this = os.getenv('WSGI_ALT_VIRTUALENV_ACTIVATE_THIS')
if not activate_this:
    raise Exception('WSGI_ALT_VIRTUALENV_ACTIVATE_THIS is not set')

def get_virtualenv_handler():
    log('Activating virtualenv with %s\n' % activate_this)
    execfile(activate_this, dict(__file__=activate_this))

    log('Getting handler %s\n' % os.getenv('WSGI_ALT_VIRTUALENV_HANDLER'))
    handler = get_wsgi_handler(os.getenv('WSGI_ALT_VIRTUALENV_HANDLER'))
    log('Got handler: %r\n' % handler)
    return handler

def get_venv_handler():
    log('Activating venv with executable at %s\n' % activate_this)
    import site
    sys.executable = activate_this
    old_sys_path, sys.path = sys.path, []

    site.main()

    sys.path.insert(0, '')
    for item in old_sys_path:
        if item not in sys.path:
            sys.path.append(item)

    log('Getting handler %s\n' % os.getenv('WSGI_ALT_VIRTUALENV_HANDLER'))
    handler = get_wsgi_handler(os.getenv('WSGI_ALT_VIRTUALENV_HANDLER'))
    log('Got handler: %r\n' % handler)
    return handler

```

Customize Git deployment

Azure will determine that your application uses Python **if both of these conditions are true:**

- requirements.txt file in the root folder
- any .py file in the root folder OR a runtime.txt that specifies python

When that's the case, it will use a Python specific deployment script, which performs the standard synchronization of files, as well as additional Python operations such as:

- Automatic management of virtual environment
- Installation of packages listed in requirements.txt using pip
- Creation of the appropriate web.config based on the selected Python version.
- Collect static files for Django applications

You can control certain aspects of the default deployment steps without having to customize the script.

If you want to skip all Python specific deployment steps, you can create this empty file:

```
\.skipPythonDeployment
```

For more control over deployment, you can override the default deployment script by creating the following files:

```
\.deployment  
\deploy.cmd
```

You can use the [Azure command-line interface](#) to create the files. Use this command from your project folder:

```
azure site deploymentscript --python
```

When these files don't exist, Azure creates a temporary deployment script and runs it. It is identical to the one you create with the command above.

Troubleshooting - Package Installation

Some packages may not install using pip when run on Azure. It may simply be that the package is not available on the Python Package Index. It could be that a compiler is required (a compiler is not available on the machine running the web app in Azure App Service).

In this section, we'll look at ways to deal with this issue.

Request wheels

If the package installation requires a compiler, you should try contacting the package owner to request that wheels be made available for the package.

With the recent availability of [Microsoft Visual C++ Compiler for Python 2.7](#), it is now easier to build packages that have native code for Python 2.7.

Build wheels (requires Windows)

Note: When using this option, make sure to compile the package using a Python environment that matches the platform/architecture/version that is used on the web app in Azure App Service (Windows/32-bit/2.7 or 3.4).

If the package doesn't install because it requires a compiler, you can install the compiler on your local machine and build a wheel for the package, which you will then include in your repository.

Mac/Linux Users: If you don't have access to a Windows machine, see [Create a Virtual Machine Running Windows](#) for how to create a VM on Azure. You can use it to build the wheels, add them to the repository, and discard the VM if you like.

For Python 2.7, you can install [Microsoft Visual C++ Compiler for Python 2.7](#).

For Python 3.4, you can install [Microsoft Visual C++ 2010 Express](#).

To build wheels, you'll need the wheel package:

```
env\scripts\pip install wheel
```

You'll use `pip wheel` to compile a dependency:

```
env\scripts\pip wheel azure==0.8.4
```

This creates a .whl file in the \wheelhouse folder. Add the \wheelhouse folder and wheel files to your repository.

Edit your requirements.txt to add the `--find-links` option at the top. This tells pip to look for an exact match in the local folder before going to the python package index.

```
--find-links wheelhouse  
azure==0.8.4
```

If you want to include all your dependencies in the \wheelhouse folder and not use the python package index at all, you can force pip to ignore the package index by adding `--no-index` to the top of your requirements.txt.

```
--no-index
```

Customize installation

You can customize the deployment script to install a package in the virtual environment using an alternate installer, such as easy_install. See deploy.cmd for an example that is commented out. Make sure that such packages aren't listed in requirements.txt, to prevent pip from installing them.

Add this to the deployment script:

```
env\scripts\easy_install somepackage
```

You may also be able to use easy_install to install from an exe installer (some are zip compatible, so easy_install supports them). Add the installer to your repository, and invoke easy_install by passing the path to the executable.

Add this to the deployment script:

```
env\scripts\easy_install "%DEPLOYMENT_SOURCE%\installers\somepackage.exe"
```

Include the virtual environment in the repository (requires Windows)

Note: When using this option, make sure to use a virtual environment that matches the platform/architecture/version that is used on the web app in Azure App Service (Windows/32-bit/2.7 or 3.4).

If you include the virtual environment in the repository, you can prevent the deployment script from doing virtual environment management on Azure by creating an empty file:

```
.skipPythonDeployment
```

We recommend that you delete the existing virtual environment on the app, to prevent leftover files from when the virtual environment was managed automatically.

Troubleshooting - Virtual Environment

The deployment script will skip creation of the virtual environment on Azure if it detects that a compatible virtual

environment already exists. This can speed up deployment considerably. Packages that are already installed will be skipped by pip.

In certain situations, you may want to force delete that virtual environment. You'll want to do this if you decide to include a virtual environment as part of your repository. You may also want to do this if you need to get rid of certain packages, or test changes to requirements.txt.

There are a few options to manage the existing virtual environment on Azure:

Option 1: Use FTP

With an FTP client, connect to the server and you'll be able to delete the env folder. Note that some FTP clients (such as web browsers) may be read-only and won't allow you to delete folders, so you'll want to make sure to use an FTP client with that capability. The FTP host name and user are displayed in your web app's blade on the [Azure Portal](#).

Option 2: Toggle runtime

Here's an alternative that takes advantage of the fact that the deployment script will delete the env folder when it doesn't match the desired version of Python. This will effectively delete the existing environment, and create a new one.

1. Switch to a different version of Python (via runtime.txt or the **Application Settings** blade in the Azure Portal)
2. git push some changes (ignore any pip install errors if any)
3. Switch back to initial version of Python
4. git push some changes again

Option 3: Customize deployment script

If you've customized the deployment script, you can change the code in deploy.cmd to force it to delete the env folder.

Next steps

For more information, see the [Python Developer Center](#).

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Integrate your app with an Azure Virtual Network

12/18/2017 • 22 min to read • [Edit Online](#)

This document describes the Azure App Service virtual network integration feature and shows how to set it up with apps in [Azure App Service](#). If you are unfamiliar with Azure Virtual Networks (VNets), this is a capability that allows you to place many of your Azure resources in a non-internet routeable network that you control access to. These networks can then be connected to your on-premises networks using a variety of VPN technologies. To learn more about Azure Virtual Networks, start with the information here: [Azure Virtual Network Overview](#).

The Azure App Service has two forms.

1. The multi-tenant systems that support the full range of pricing plans
2. The App Service Environment (ASE) premium feature, which deploys into your VNet.

This document goes through VNet Integration and not App Service Environment. If you want to learn more about the ASE feature, start with the information here: [App Service Environment introduction](#).

VNet Integration gives your web app access to resources in your virtual network but does not grant 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. Private site access is only available with an ASE configured with an Internal Load Balancer (ILB). For details on using an ILB ASE, start with the article here: [Creating and using an ILB ASE](#).

A common scenario where you would use VNet Integration is enabling access from your web app to a database or a web service running on a virtual machine in your Azure virtual network. With VNet Integration, you don't need to expose a public endpoint for applications on your VM but can use the private non-internet routable addresses instead.

The VNet Integration feature:

- requires a Standard, Premium, or Isolated pricing plan
- works with Classic or Resource Manager VNet
- supports TCP and UDP
- works with Web, Mobile, API apps and Function apps
- enables an app to connect to only 1 VNet at a time
- enables up to five VNets to be integrated with in an App Service Plan
- allows the same VNet to be used by multiple apps in an App Service Plan
- supports a 99.9% SLA due to the SLA on the VNet Gateway

There are some things that VNet Integration does not support including:

- mounting a drive
- AD integration
- NetBios
- private site access

Getting started

Here are some things to keep in mind before connecting your web app to a virtual network:

- VNet Integration only works with apps in a **Standard, Premium, or Isolated** pricing plan. If you enable the feature, and then scale your App Service Plan to an unsupported pricing plan your apps lose their connections to the VNets they are using.

- 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 configured with Static routing, you cannot enable point-to-site Virtual Private Network (VPN).
- The VNet must be in the same subscription as your App Service Plan(ASP).
- The apps that integrate with a VNet use the DNS that is specified for that VNet.
- By default your integrating apps only route traffic into your VNet based on the routes that are defined in your VNet.

Enabling VNet Integration

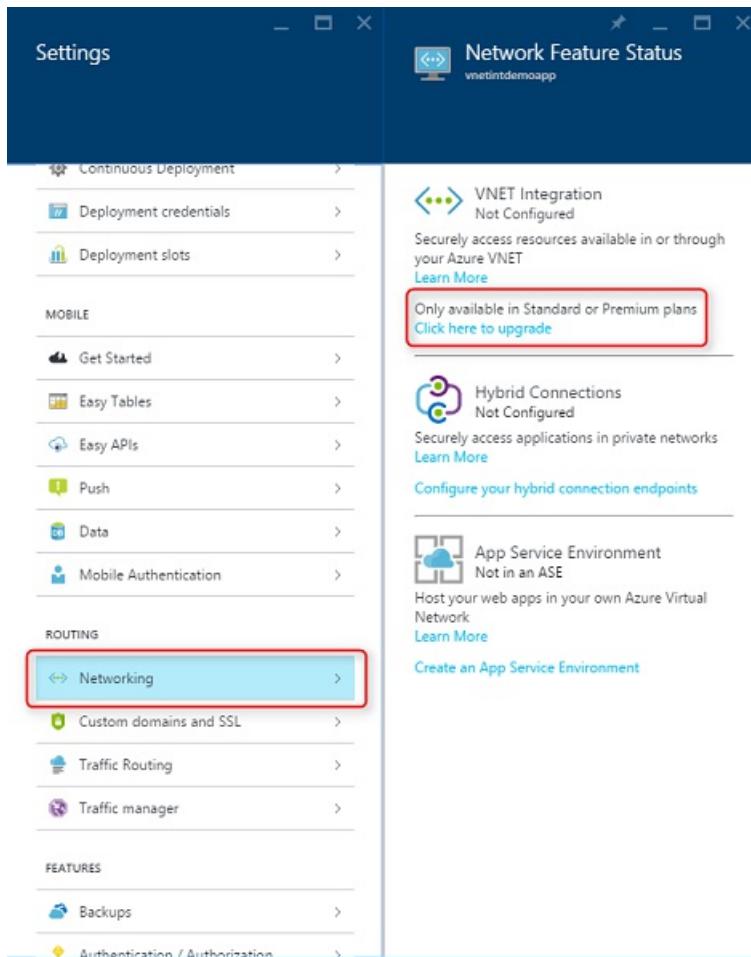
You have the option to connect your app to a new or existing virtual network. If you create a new network as a part of your integration, then in addition to just creating the VNet, a dynamic routing gateway is pre-configured for you and Point to Site VPN is enabled.

NOTE

Configuring a new virtual network integration can take several minutes.

To enable VNet Integration, open your app Settings and then select Networking. The UI that opens up offers three networking choices. This guide is only going into VNet Integration though Hybrid Connections and App Service Environments are discussed later in this document.

If your app is not in the correct pricing plan, the UI enables you to scale your plan to a higher pricing plan of your choice.

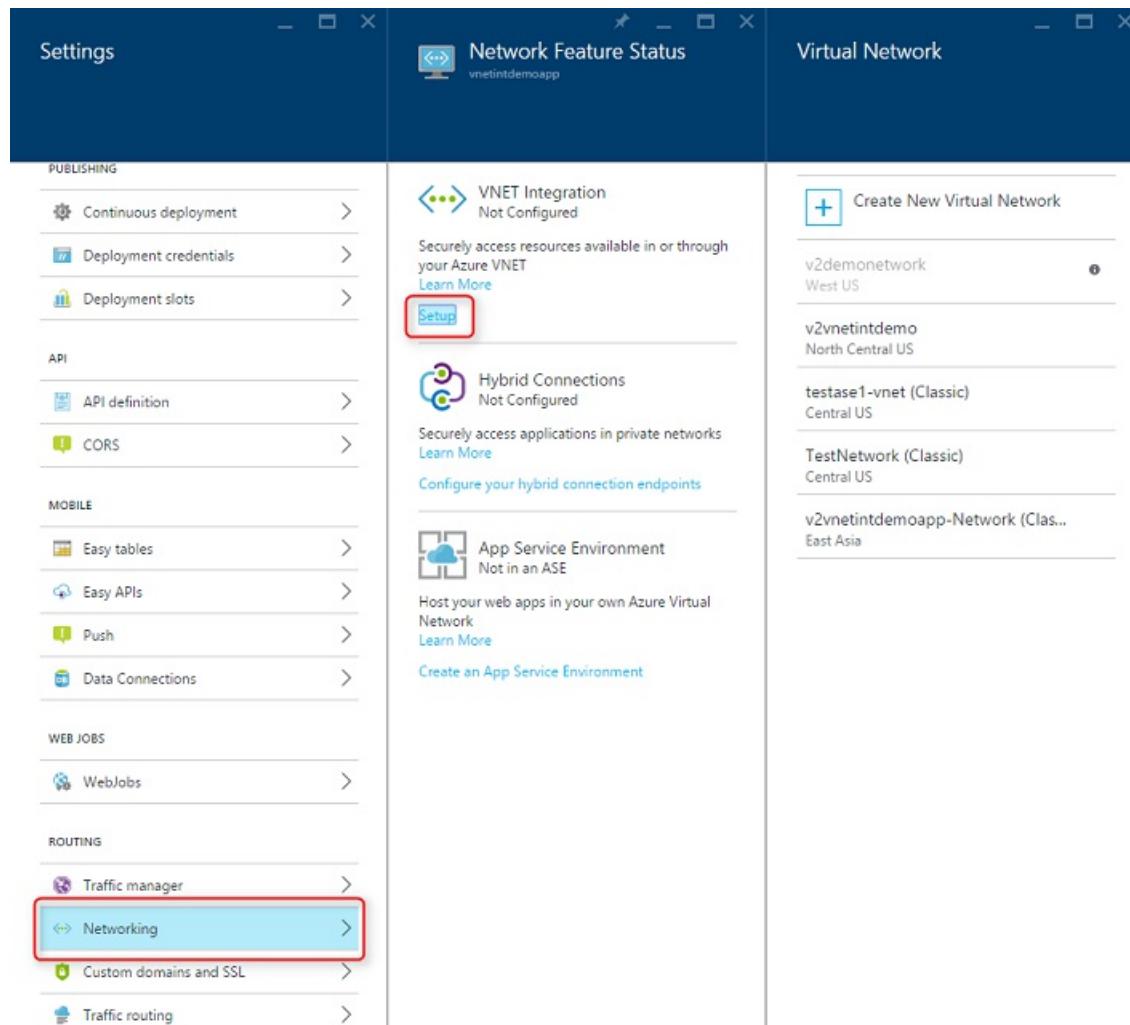


Enabling VNet Integration with a pre-existing VNet

The VNet Integration UI allows you to select from a list of your VNets. The Classic VNets indicate that they are such with the word "Classic" in parentheses next to the VNet name. The list is sorted such that the Resource Manager

VNets are listed first. In the image shown below you can see that only one VNet can be selected. There are multiple reasons that a VNet can be grayed out including:

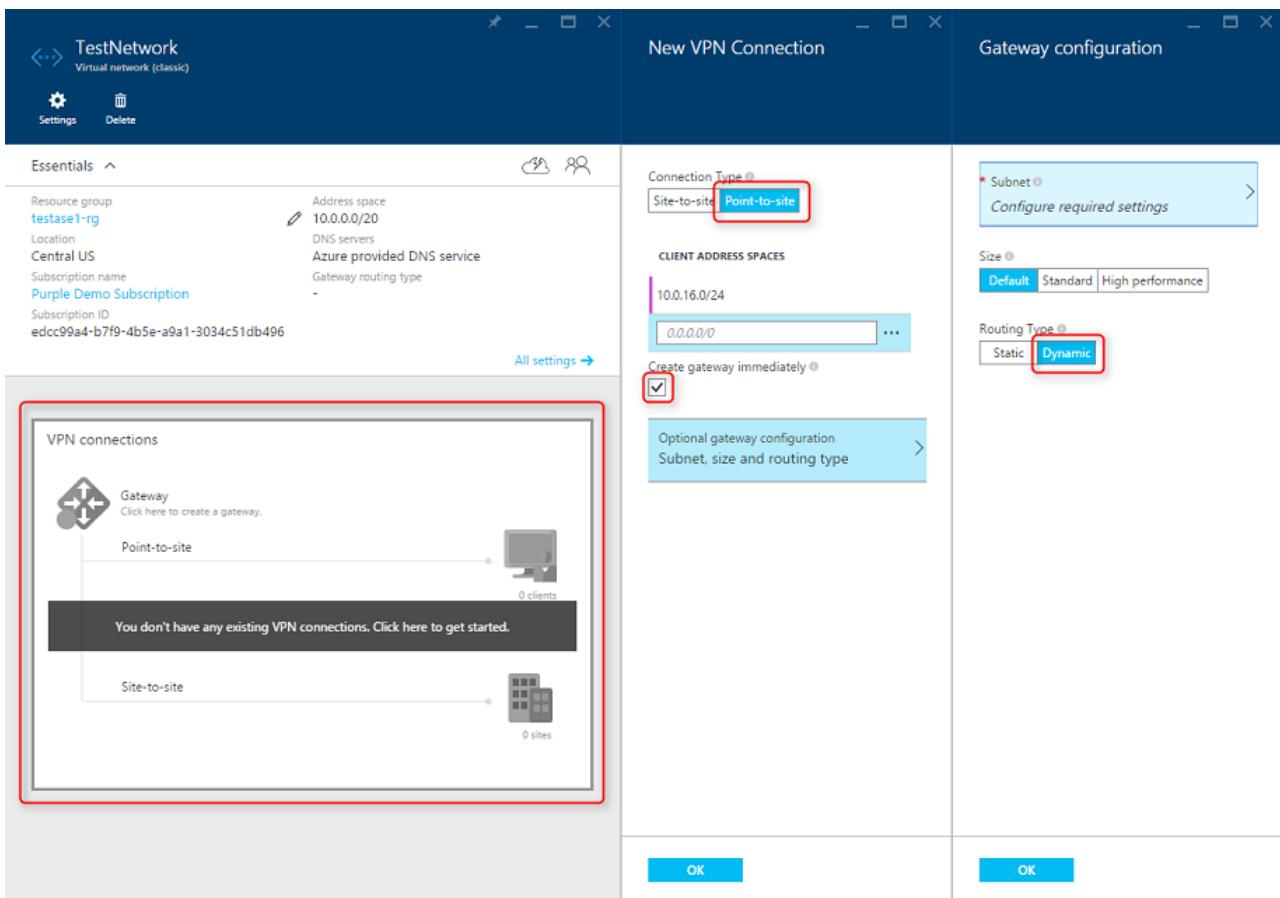
- the VNet is in another subscription that your account has access to
- the VNet does not have Point to Site enabled
- the VNet does not have a dynamic routing gateway



To enable integration simply click on the VNet you wish to integrate with. After you select the VNet, your app is automatically restarted for the changes to take effect.

Enable Point to Site in a Classic VNet

If your VNet does not have a gateway nor has Point to Site, then you have to set that up first. To do this for a Classic VNet, go to the [Azure portal](#) and bring up the list of Virtual Networks(classic). From here, click on the network you want to integrate with and click on the big box under Essentials called VPN Connections. From here, you can create your point to site VPN and even have it create a gateway. After you go through the point to site with gateway creation experience it is about 30 minutes before it is ready.



Enabling Point to Site in a Resource Manager VNet

To configure a Resource Manager VNet with a gateway and Point to Site, you can use either PowerShell as documented here, [Configure a Point-to-Site connection to a virtual network using PowerShell](#) or use the Azure portal as documented here, [Configure a Point-to-Site connection to a VNet using the Azure portal](#). The UI to perform this capability is not yet available. Note that you don't need to create certificates for the Point to Site configuration. This is automatically configured when you connect your WebApp to the VNet.

Creating a pre-configured VNet

If you want to create a new VNet that is configured with a gateway and Point-to-Site, then the App Service networking UI has the capability to do that but only for a Resource Manager VNet. If you wish to create a Classic VNet with a gateway and Point-to-Site, then you need to do this manually through the Networking user interface.

To create a Resource Manager VNet through the VNet Integration UI, simply select **Create New Virtual Network** and provide the:

- Virtual Network Name
- Virtual Network Address Block
- Subnet Name
- Subnet Address Block
- Gateway Address Block
- Point-to-Site Address Block

If you want this VNet to connect to any other networks, then you should avoid picking IP address space that overlaps with those networks.

NOTE

Resource Manager VNet creation with a gateway takes about 30 minutes and currently does not integrate the VNet with your app. After your VNet is created with the gateway you need to come back to your app VNet Integration UI and select your new VNet.

The screenshot shows three windows side-by-side:

- Network Feature Status**: Shows VNET Integration (Not Configured) and App Service Environment (Not in an ASE).
- Virtual Network**: Lists existing VNets: v2demonetwork (West US), v2vnetintdemo (North Central US), testase1-vnet (Classic) (Central US), and v2vnetintdemoapp-Network (Classic) (East Asia).
- Create New Virtual Network**: A form to create a new VNet. It includes fields for "Virtual Network" (with a note about CIDR notation), "Virtual Network Address Block" (10.0.0.0/8, Range: 10.0.0.0 - 10.255.255.255), "Subnet Name" (default), "Subnet Address Block" (10.1.0.0/16, Range: 10.1.0.0 - 10.1.255.255), "Gateway Subnet Address Block" (10.2.0.0/16, Range: 10.2.0.0 - 10.2.255.255), and "Point-to-Site Address Block" (172.16.0.0/16, Range: 172.16.0.0 - 172.16.255.255). It also includes a note about VNET selection.

Azure VNets normally are created within private network addresses. By default the VNet Integration feature routes any traffic destined for those IP address ranges into your VNet. The private IP address ranges are:

- 10.0.0.0/8 - this is the same as 10.0.0.0 - 10.255.255.255
- 172.16.0.0/12 - this is the same as 172.16.0.0 - 172.31.255.255
- 192.168.0.0/16 - this is the same as 192.168.0.0 - 192.168.255.255

The VNet address space needs to be specified in CIDR notation. If you are unfamiliar with CIDR notation, it is a method for specifying address blocks using an IP address and an integer that represents the network mask. As a quick reference, consider that 10.1.0.0/24 would be 256 addresses and 10.1.0.0/25 would be 128 addresses. An IPv4 address with a /32 would be just 1 address.

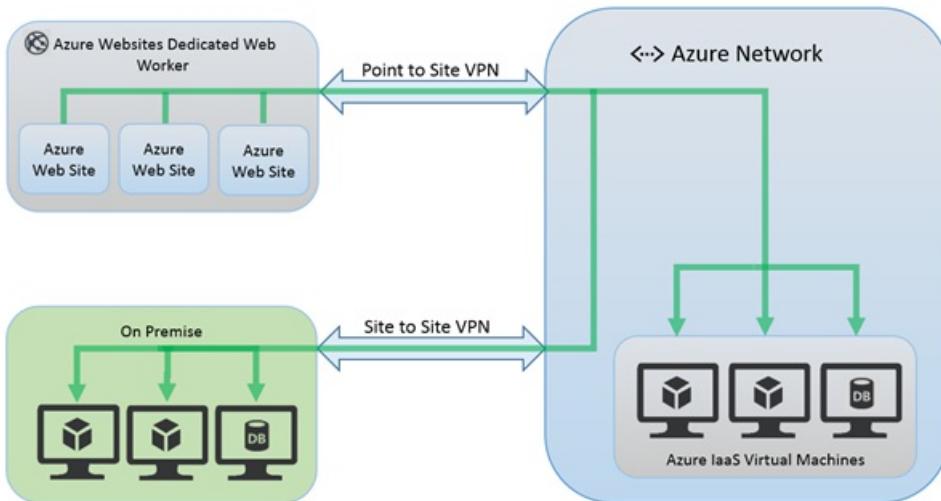
If you set the DNS server information here, then that is set for your VNet. After VNet creation you can edit this information from the VNet user experiences. If you change the DNS of the VNet, then you need to perform a Sync Network operation.

When you create a Classic VNet using the VNet Integration UI, it creates a VNet in the same resource group as your app.

How the system works

Under the covers this feature builds on top of Point-to-Site VPN technology to connect your app to your VNet. Apps in Azure App Service have a multi-tenant system architecture, which precludes provisioning an app directly in a VNet as is done with virtual machines. By building on point-to-site technology we limit network access to just the

virtual machine hosting the app. Access to the network is further restricted on those app hosts so that your apps can only access the networks that you configure them to access.



If you haven't configured a DNS server with your virtual network, your app will need to use IP addresses to reach resource in the VNet. While using IP addresses, remember that the major benefit of this feature is that it enables you to use the private addresses within your private network. If you set your app up to use public IP addresses for one of your VMs, then you aren't using the VNet Integration feature and are communicating over the internet.

Managing the VNet Integrations

The ability to connect and disconnect to a VNet is at an app level. Operations that can affect the VNet Integration across multiple apps are at an ASP level. From the UI that is shown at the app level, you can get details on your VNet. Most of the same information is also shown at the ASP level.

The screenshot shows two adjacent windows from the Azure portal:

- Network Feature Status:** This window is for the app 'vnetintdemoapp'. It displays:
 - VNET Integration:** Connected to 'testase1-vnet'. It includes a note: "Securely access resources available in or through your Azure VNET". A link 'Learn More' and a button 'Click here to configure' are present.
 - Hybrid Connections:** Not Configured. It includes a note: "Securely access applications in private networks". A link 'Learn More' and a button 'Configure your hybrid connection endpoints' are present.
 - App Service Environment:** Not in an ASE. It includes a note: "Host your web apps in your own Azure Virtual Network". A link 'Learn More' and a button 'Create an App Service Environment' are present.
- Virtual Network Integration:** This window is also for the app 'vnetintdemoapp'. It displays:
 - VNET NAME:** testase1-vnet
 - LOCATION:** Central US
 - CERTIFICATE STATUS:** Certificates in sync
 - GATEWAY STATUS:** Online
 - VNET ADDRESS SPACE:** Start Address: 10.254.0.0, End Address: 10.254.15.255
 - POINT-TO-SITE ADDRESS SPACE:** Start Address: 10.0.0.0, End Address: 10.0.0.255
 - SITE-TO-SITE ADDRESS SPACE:** Note: "Site-to-site is not configured on this VNET."
 - DNS SERVERS:** Note: "No DNS servers are configured on this VNET."
 - IP ADDRESSES ROUTED TO VNET:** Start Address: 10.0.0.0, End Address: 10.255.255.255

From the Network Feature Status page, you can see if your app is connected to your VNet. If your VNet gateway is down for whatever reason, then this would show as not-connected.

The information you now have available to you in the app level VNet Integration UI is the same as the detail information you get from the ASP. Here are those items:

- VNet Name - This link opens the Azure virtual network UI
- Location - This reflects the location of your VNet. It is possible to integrate with a VNet in another location.
- Certificate Status - There are certificates used to secure the VPN connection between the app and the VNet. This reflects a test to ensure they are in sync.
- Gateway Status - Should your gateways be down for whatever reason then your app cannot access resources in the VNet.
- VNet address space - This is the IP address space for your VNet.
- Point to Site address space - This is the point to site IP address space for your VNet. Your app shows communication as coming from one of the IPs in this address space.
- Site to site address space - You can use Site to Site VPNs to connect your VNet to your on-premises resources or to other VNet. Should you have that configured then the IP ranges defined with that VPN connection shows here.
- DNS Servers - If you have DNS Servers configured with your VNet, then they are listed here.
- IPs routed to the VNet - There are a list of IP addresses that your VNet has routing defined for, and those addresses show here.

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 do this simply click Disconnect at the top. This action does not change your VNet. The VNet and its configuration including the gateways remains unchanged. If you then want to delete your VNet, you need to first delete the resources in it including the gateways.

The App Service Plan view has a number of additional operations. It is also accessed differently than from the app. To reach the ASP Networking UI simply open your ASP UI and scroll down. There is a UI element called Network Feature Status. It gives some minor details around your VNet Integration. Clicking on this UI opens the Network Feature Status UI. If you then click on "Click here to manage", the UI that lists the VNet Integrations in this ASP opens.

The screenshot shows three windows side-by-side:

- hcdemoplan** App Service Plan: Shows basic info like Resource group (hcdemo-rg), Location (North Central US), and a chart of CPU and Memory usage over time.
- Network Feature Status**: A summary card for VNet integration, stating 2 VNets configured and linking to "Click here to manage".
- Virtual Network Integration**: A detailed table listing VNet integrations. It shows:

VNET NAME	GATEWAY STATUS	LOCATION
hcdemoplannet	Online	East Asia
testas1-vnet	Online	Central US

The location of the ASP is good to remember when looking at the locations of the VNets you are integrating with. When the VNet is in another location you are far more likely to see latency issues.

The VNets integrated with is a reminder on how many VNets your apps are integrated with in this ASP and how

many you can have.

To see added details on each VNet, just click on the VNet you are interested in. In addition to the details that were noted earlier, you can also see a list of the apps in this ASP that are using that VNet.

With respect to actions there are two primary actions. The first is the ability to add routes that drive traffic leaving your app into your VNet. The second action is the ability to sync certificates and network information.

The screenshot displays two windows of the 'Virtual Network Integration' tool. The left window is for the App Service Plan 'hcdemoplan' and shows the following details:

App Service Plan	hcdemoplan
App Service Plan Location	North Central US
VNETs integrated with	2 out of 3 possible

The right window is for the VNet 'testase1-vnet' and shows the following configuration:

VNET NAME	testase1-vnet
LOCATION	Central US
CERTIFICATE STATUS	Certificates in sync
GATEWAY STATUS	Online

Below these settings, there are sections for 'APPS USING VIRTUAL NETWORK' (listing 'vnetintdemoapp') and 'VNET ADDRESS SPACE' (listing 'START ADDRESS' 10.254.0.0 and 'END ADDRESS' 10.254.15.255). There are also sections for 'POINT-TO-SITE ADDRESS SPACE' (listing 'START ADDRESS' 10.0.0.0 and 'END ADDRESS' 10.0.0.255) and 'SITE-TO-SITE ADDRESS SPACE' (not configured). The 'DNS SERVERS' section indicates no DNS servers are configured. Finally, the 'IP ADDRESSES ROUTED TO VNET' section lists ranges: 10.0.0.0 to 10.255.255, 172.16.0.0 to 172.31.255.255, and 192.168.0.0 to 192.168.255.255.

Routing As noted earlier the routes that are defined in your VNet are what is used for directing traffic into your VNet from your app. There are some uses though where customers want to send additional outbound traffic from an app into the VNet and for them this capability is provided. What happens to the traffic after that is up to how the customer configures their VNet.

Certificates The Certificate Status reflects a check being performed by the App Service to validate that the certificates that we are using for the VPN connection are still good. When VNet Integration enabled, then if this is the first integration to that VNet from any apps in this ASP, there is a required exchange of certificates to ensure the security of the connection. Along with the certificates we get the DNS configuration, routes and other similar things that describe the network. If those certificates or network information is changed, then you need to click "Sync Network". **NOTE:** When you click "Sync Network" then you cause a brief outage in connectivity between your app and your VNet. While your app is not restarted, the loss of connectivity could cause your site to not function properly.

Accessing on-premises resources

One of the benefits of the VNet Integration feature is that if your VNet is connected to your on-premises network with a Site to Site VPN then your apps can have access to your on-premises resources from your app. For this to work though you may need to update your on-premises VPN gateway with the routes for your Point to Site IP range. When the Site to Site VPN is first set up then the scripts used to configure it should set up routes including your Point to Site VPN. If you add the Point to Site VPN after you create your Site to Site VPN, then you need to update the routes manually. Details on how to do that vary per gateway and are not described here.

NOTE

The VNet Integration feature does not integrate an app with a VNet that has an ExpressRoute Gateway. Even if the ExpressRoute Gateway is configured in [coexistence mode](#) the VNet Integration does not work. If you need to access resources through an ExpressRoute connection, then you can use an [App Service Environment](#), which runs in your VNet.

Pricing details

There are a few pricing nuances that you should be aware of when using the VNet Integration feature. There are 3 related charges to the use of this feature:

- ASP pricing tier requirements
- Data transfer costs
- VPN Gateway costs.

For your apps to be able to use this feature, they need to be in a Standard or Premium App Service Plan. You can see more details on those costs here: [App Service Pricing](#).

Due to the way Point to Site VPNs are handled, you always have a charge for outbound data through your VNet Integration connection even if the VNet is in the same data center. To see what those charges are, take a look here: [Data Transfer Pricing Details](#).

The last item is the cost of the VNet gateways. If you don't need the gateways for something else such as Site to Site VPNs, then you are paying for gateways to support the VNet Integration feature. There are details on those costs here: [VPN Gateway Pricing](#).

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 console experiences you can use. One is from the Kudu console and the other is the console that you can reach in the Azure portal. To get to the Kudu console from your app go to Tools -> Kudu. This is the same as going to [sitename].scm.azurewebsites.net. Once that opens simply 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 there have been 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.

The next tool allows you to test for TCP connectivity to a host and port combination. This tool is called tcpping.exe 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:

- **There is a firewall in the way** If you have a firewall in the way, you will hit the TCP timeout. That 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 is not 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 does not use the DNS your VNet is configured with.
- **Invalid P2S IP range** The point to site IP range needs to be in the RFC 1918 private IP 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). If the range uses IPs outside of that, then things won't work.

If those items don't answer your problem, look first for the simple things like:

- Does the Gateway show as being up in the portal?
- Do certificates show as being in sync?
- Did anybody change the network configuration without doing a "Sync Network" in the affected ASPs?

If your gateway is down, then bring it back up. If your certificates are out of sync, then go to the ASP view of your VNet Integration and hit "Sync Network". If you suspect that there has been a change made to your VNet configuration and it wasn't sync'd with your ASPs, then go to the ASP view of your VNet Integration and hit "Sync Network". Just as a reminder, this causes a brief outage with your VNet connection and your apps.

If all of that is fine, then you need to dig in a bit deeper:

- Are there any other apps using VNet Integration to reach resources in the same VNet?
- Can you go to the app console and use tcpping to reach any other resources in your VNet?

If either of the above are true, then your VNet Integration is fine and the problem is somewhere else. This is where it gets to be more of a challenge because there is no simple way to see why you can't reach a host:port. 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 check this by going onto that host and using "netstat -aon" from the cmd prompt. This shows you what process ID is listening on what port.
- 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 IP in your Point to Site IP range that your app will use so you need to allow access from the entire range.

Additional debug steps include:

- log onto another VM in your VNet and attempt to reach your resource host:port from there. There are some TCP ping utilities that you can use for this purpose or can even use telnet if need be. The purpose here is just to determine if connectivity is there from this other VM.
- bring up an application on another VM and test access to that host and port from the console from your app

On-premises resources

If your app cannot reach resources on-premises, then the first thing you should check is if you can reach a resource in your VNet. If that works, try to reach the on-premises application from a VM in the VNet. You can use telnet or a TCP ping utility. If your VM can't reach your on-premises resource, then make sure your Site to Site VPN connection is working. If it is working, then check the same things noted earlier as well as the on-premises gateway configuration and status.

Now if your VNet hosted VM can reach your on-premises system but your app can't then the reason is likely one of the following:

- your routes are not configured with your point to site IP 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 have a User Defined Route(UDR) in your VNet that prevents your Point to Site based traffic from reaching your on-premises network

PowerShell automation

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](#).

Hybrid Connections and App Service Environments

There are three features that enable access to VNet hosted resources. They are:

- VNet Integration
- Hybrid Connections
- App Service Environments

Hybrid Connections requires you to install a relay agent called the Hybrid Connection Manager(HCM) in your network. The HCM needs to be able to connect to Azure and also to your application. This solution is especially great from a remote network such as your on-premises network or even another cloud hosted network because it does not require an internet accessible endpoint. The HCM only runs on Windows and you can have up to five instances running to provide high availability. Hybrid Connections only supports TCP though and each HC endpoint has to match to a specific host:port combination.

The App Service Environment feature allows you to run an instance of the Azure App Service in your VNet. This lets your apps access resources in your VNet without any extra steps. Some of the other benefits of an App Service Environment are that you can use Dv2 based workers with up to 14 GB of RAM. Another benefit is that you can scale the system to meet your needs. Unlike the multi-tenant environments where your ASP is limited to 20 instances, in an ASE you can scale up to 100 ASP instances. One of the things you get with an ASE that you don't get with VNet Integration is that an App Service Environment can work with an ExpressRoute VPN.

While there is some use case overlap, none of these feature can replace any of the others. Knowing what feature to use is tied to your needs. For example:

- If you are a developer and simply want to run a site in Azure and have it access the database on the workstation under your desk, then the easiest thing to use is Hybrid Connections.
- If you are a large organization that wants to put a large number of web properties in the public cloud and manage them in your own network, then you want to go with the App Service Environment.

- If you have a number of App Service hosted apps and simply want to access resources in your VNet, then VNet Integration is the way to go.

Beyond the use cases, there are some simplicity related aspects. If your VNet is already connected to your on-premises network, then using VNet Integration or an App Service Environment is an easy way to consume on-premises resources. On the other hand, if your VNet is not connected to your on-premises network then it's a lot more overhead to set up a site to site VPN with your VNet compared with installing the HCM.

Beyond the functional differences, there are also pricing differences. The App Service Environment feature is a Premium service offering but offers the most network configuration possibilities in addition to other great features. VNet Integration can be used with Standard or Premium ASPs and is perfect for securely consuming resources in your VNet from the multi-tenant App Service. Hybrid Connections currently depends on a BizTalk account, which has pricing levels that start free and then get progressively more expensive based on the amount you need. When it comes to working across many networks though, there is no other feature like Hybrid Connections, which can enable you to access resources in well over 100 separate networks.

Deploy your app to Azure App Service with a ZIP file

12/14/2017 • 4 min to read • [Edit Online](#)

This article shows you how to use a ZIP 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.

For more information, see [Kudu documentation](#).

Prerequisites

To complete the steps in this article:

- [Create an App Service app](#), or use an app that you created for another tutorial.

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`.

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
```

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

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account. Just click the **Copy** to copy the code, paste it into the Cloud Shell, and then press enter to run it. There are two ways to launch the Cloud Shell:

Click **Try It** in the upper right corner of a code block.



Click the **Cloud Shell** button on the menu in the upper right of the [Azure portal](#).



Upload ZIP file to Cloud Shell

If you choose to run the Azure CLI from your local terminal instead, skip this step.

Follow the steps here to upload your ZIP file to the Cloud Shell.

In the [Azure portal](#), click **Resource groups** > **cloud-shell-storage-<your_region>** > **<storage_account_name>**.

In the **Overview** page of the storage account, select **Files**.

Select the automatically generated file share and select **Upload**. This file share is mounted in the Cloud Shell as `clouddrive`.

Click the file selector and select your ZIP file, then click **Upload**.

In the Cloud Shell, use `ls` to verify that you can see the uploaded ZIP file in the default `clouddrive` share.

```
ls clouddrive
```

For more information, see [Persist files in Azure Cloud Shell](#).

Deploy ZIP file

Deploy the uploaded ZIP file to your web app by using the [az webapp deployment source config-zip](#) command. If you choose not to use Cloud Shell, make sure your Azure CLI version is 2.0.21 or later. To see which version you have, run `az --version` command in the local terminal window.

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 myResourceGroup --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. If any additional custom build process is configured, it is run as well. For more information, see [Kudu documentation](#).

To view the list of deployments for this app, you must use the REST APIs (see next section).

Deploy by using REST APIs

You can use the [deployment service REST APIs](#) to deploy the .zip file to your app in Azure. Just 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](#).

The following example uses the cURL tool to send a request that contains the .zip file. You can run cURL from the terminal on a Mac or Linux computer or by using Bash on Windows. Replace the `<zip_file_path>` placeholder with the path to the location of your project zip file. Also replace `<app_name>` with the unique name of your app.

Replace the `<deployment_user>` placeholder with the username of your deployment credentials. When prompted by cURL, type in the password. To learn how to set deployment credentials for your app, see [Set and reset user-level credentials](#).

```
curl 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
```

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

9/19/2017 • 2 min 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.

IMPORTANT

We are continuously taking steps to improve Microsoft Azure Platform security. As part of this ongoing effort an upgrade of Web Applications is planned for Germany Central and Germany Northeast regions. During this Web Apps will be disabling the use of plain text FTP protocol for deployments. Our recommendation to our customers is to switch to FTPS for deployments. We do not expect any disruption to your service during this upgrade which is planned for 9/5. We appreciate your support in this effort.

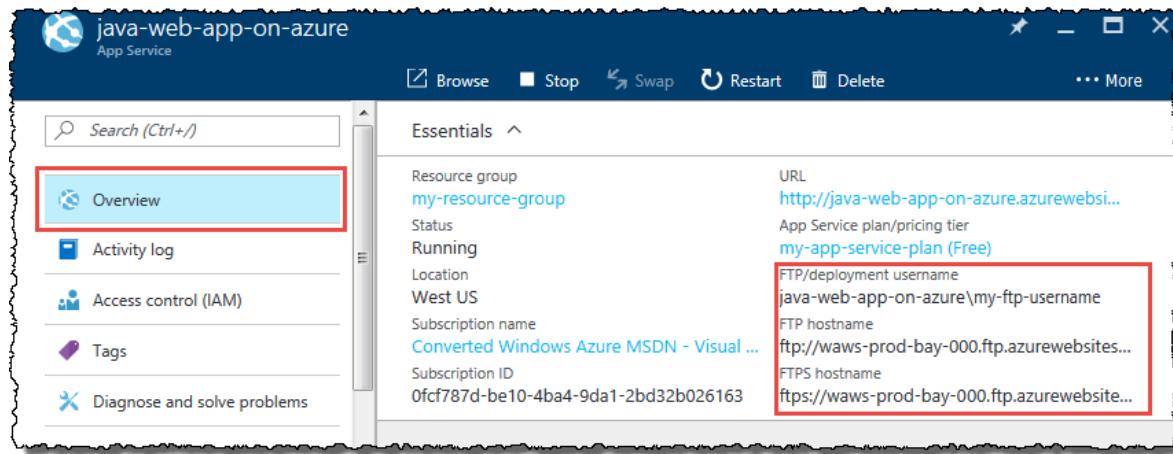
Step 1: Set deployment credentials

To access the FTP server for your app, you first need deployment credentials.

To set or reset your deployment credentials, see [Azure App Service Deployment Credentials](#). This tutorial demonstrates the use of user-level credentials.

Step 2: Get FTP connection information

1. In the [Azure portal](#), open your app's [resource blade](#).
2. Select **Overview** in the left menu, then note the values for **FTP/Deployment User**, **FTP Host Name**, and **FTPS Host Name**.



NOTE

The **FTP/Deployment User** user value as displayed by the Azure Portal including the app name in order to provide proper context for the FTP server. You can find the same information when you select **Properties** in the left menu.

Also, the deployment password is never shown. If you forget your deployment password, go back to [step 1](#) and reset your deployment password.

Step 3: Deploy files to Azure

1. From your FTP client ([Visual Studio](#), [FileZilla](#), etc), 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 restore (such as NuGet, NPM, PIP, and Composer automations)
- compilation of .NET binaries
- generation of web.config (here is a [Node.js example](#))

You must restore, build, and generate these necessary files manually on your local machine and deploy them together with your app.

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

11/7/2017 • 1 min to read • [Edit Online](#)

This tutorial shows you how to deploy to [Azure App Service](#) by syncing your content from popular cloud storage services like Dropbox and OneDrive.

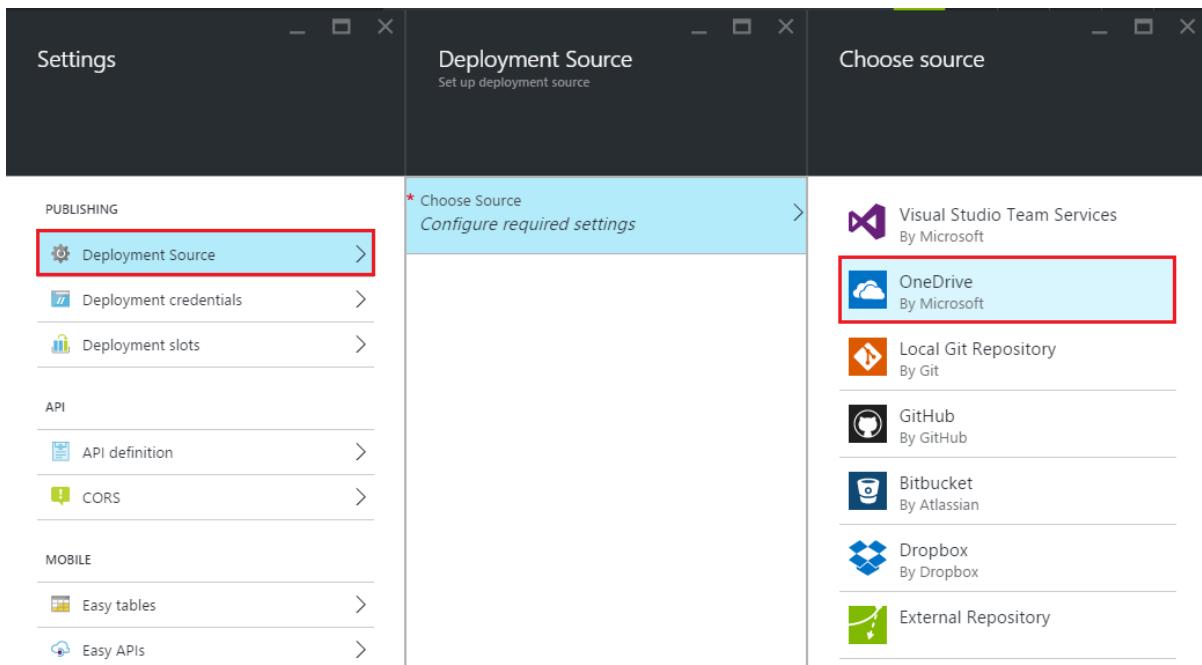
Overview of content sync deployment

The on-demand content sync deployment is powered by the [Kudu deployment engine](#) integrated with App Service. In the [Azure Portal](#), you can designate a folder in your cloud storage, work with your app code and content in that folder, and sync to App Service with the click of a button. Content sync utilizes the Kudu process for build and deployment.

How to enable content sync deployment

To enable content sync from the [Azure Portal](#), follow these steps:

1. In your app's blade in the Azure Portal, click **Settings > Deployment Source**. Click **Choose Source**, then select **OneDrive** or **Dropbox** as the source for deployment.



NOTE

Because of underlying differences in the APIs, **OneDrive for Business** is not supported at this time.

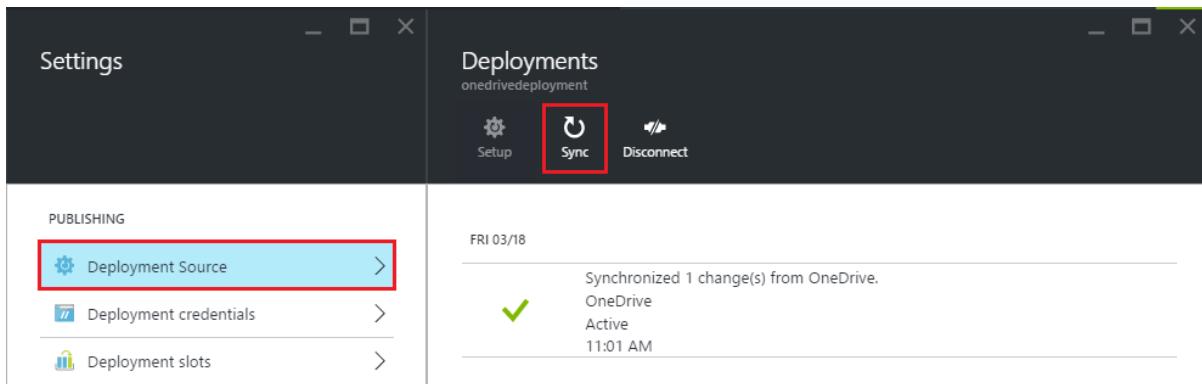
2. Complete the authorization workflow to enable App Service to access a specific pre-defined designated path for OneDrive or Dropbox where all of your App Service content will be stored.

After authorization the App Service platform will give you the option to create a content folder under the designated content path, or to choose an existing content folder under this designated content path. The designated content paths under your cloud storage accounts used for App Service sync are the following:

- **OneDrive:** Apps\Azure Web Apps

- **Dropbox:** Dropbox\Apps\Azure

3. After the initial content sync the content sync can be initiated on demand from the Azure portal. Deployment history is available with the **Deployments** blade.



More information for Dropbox deployment is available under [Deploy from Dropbox](#).

Continuous Deployment to Azure App Service

9/19/2017 • 3 min to read • [Edit Online](#)

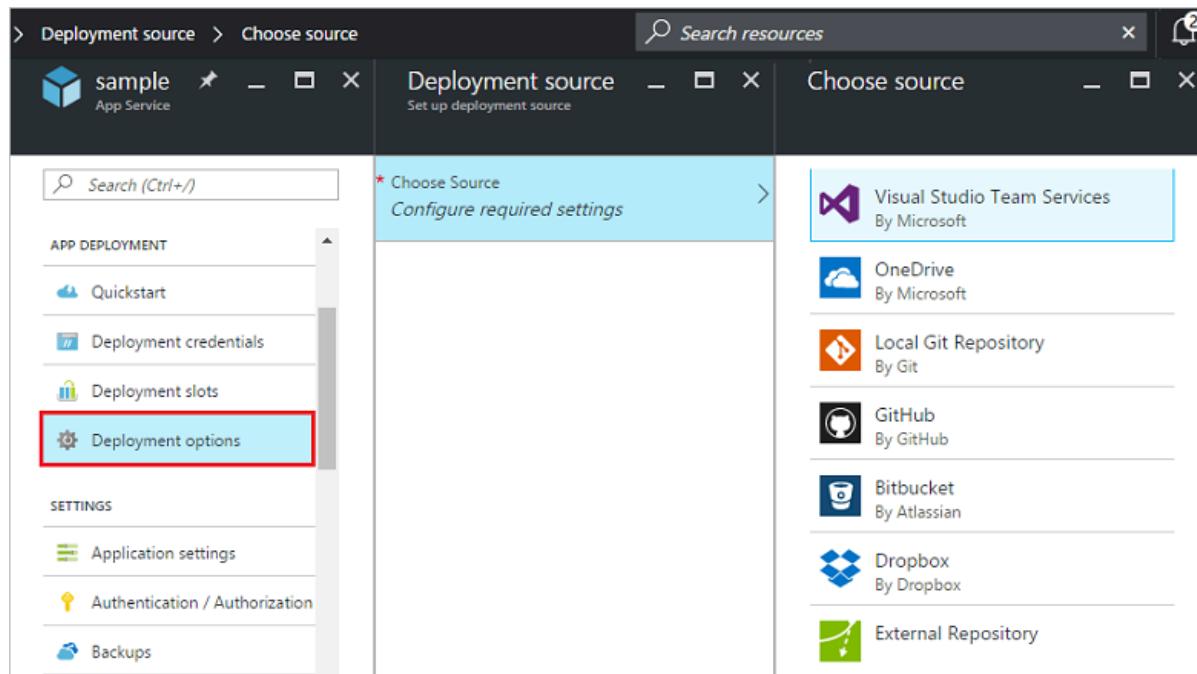
This tutorial shows you how to configure a continuous deployment workflow for your [Azure Web Apps](#). App Service integration with BitBucket, GitHub, and [Visual Studio Team Services \(VSTS\)](#) enables a continuous deployment workflow where Azure pulls in the most recent updates from your project published to one of these services. Continuous deployment is a great option for projects where multiple and frequent contributions are being integrated.

To find out how to configure continuous deployment manually from a cloud repository not listed by the Azure Portal (such as [GitLab](#)), see [Setting up continuous deployment using manual steps](#).

Enable continuous deployment

To enable continuous deployment,

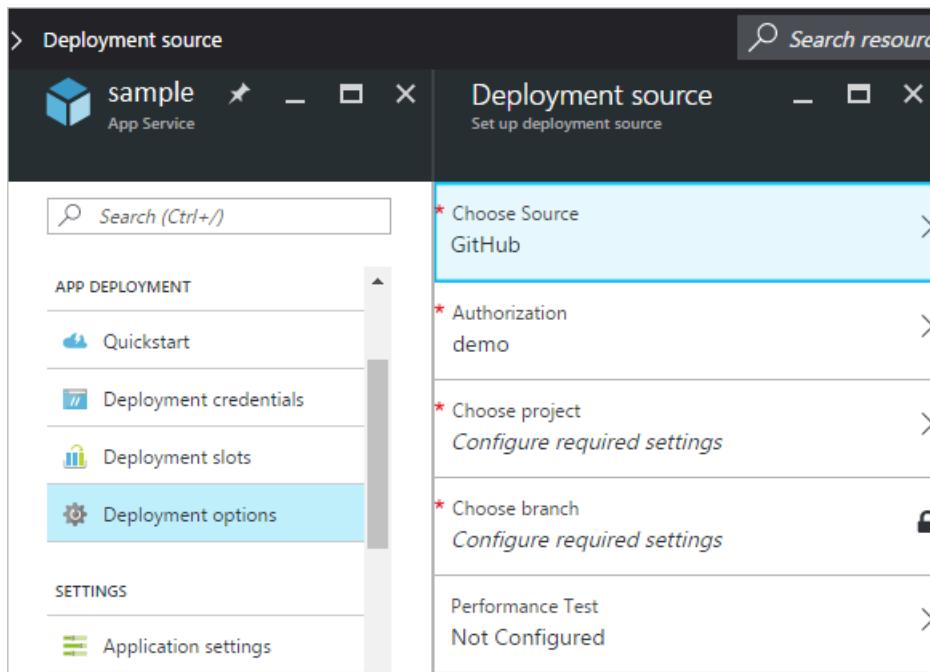
1. Publish your app content to the repository that will be used for continuous deployment.
For more information on publishing your project to these services, see [Create a repo \(GitHub\)](#), [Create a repo \(BitBucket\)](#), and [Get started with VSTS](#).
2. In your app's menu blade in the [Azure portal](#), click **APP DEPLOYMENT > Deployment options**. Click **Choose Source**, then select the deployment source.



NOTE

To configure a VSTS account for App Service deployment please see this [tutorial](#).

3. Complete the authorization workflow.
4. In the **Deployment source** blade, choose the project and branch to deploy from. When you're done, click **OK**.



NOTE

When enabling continuous deployment with GitHub or BitBucket, both public and private projects will be displayed.

App Service creates an association with the selected repository, pulls in the files from the specified branch, and maintains a clone of your repository for your App Service app. When you configure VSTS continuous deployment from the Azure portal, the integration uses the App Service [Kudu deployment engine](#), which already automates build and deployment tasks with every `git push`. You do not need to separately set up continuous deployment in VSTS. After this process completes, the **Deployment options** app blade will show an active deployment that indicates deployment has succeeded.

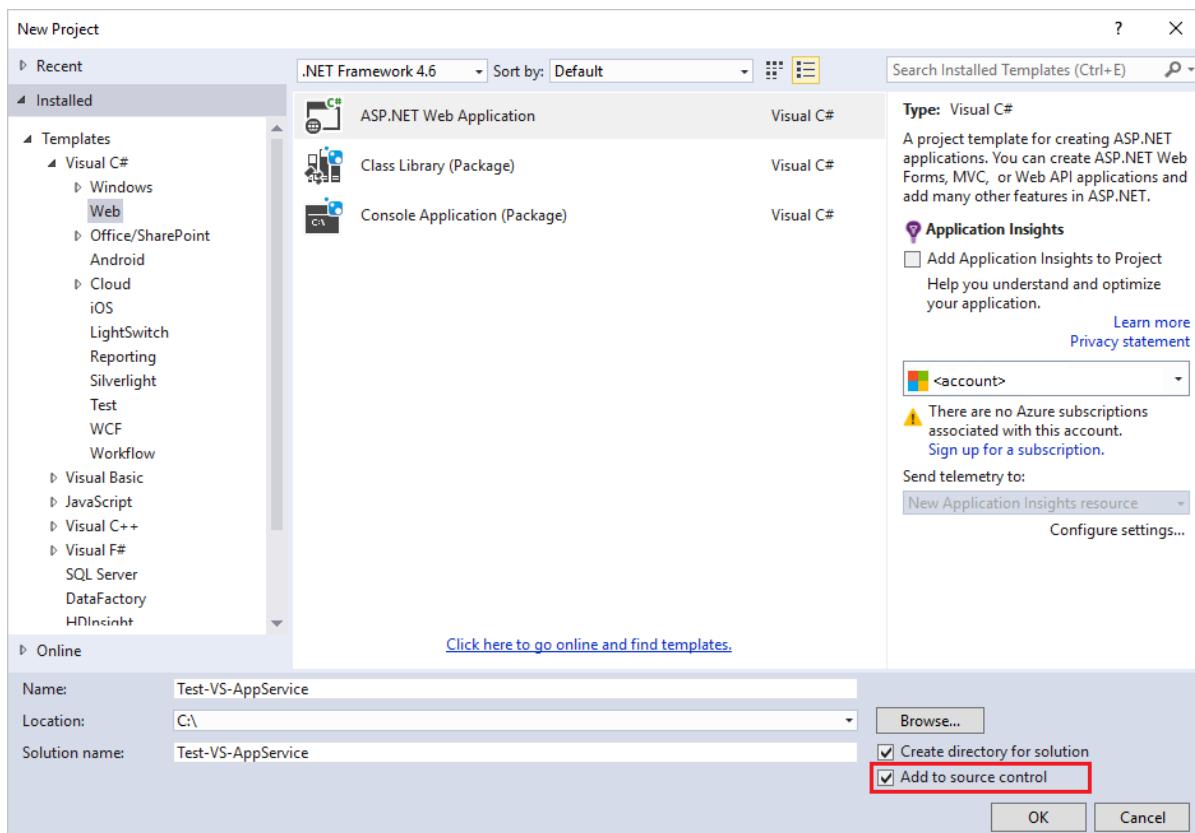
5. To verify the app is successfully deployed, click the **URL** at the top of the app's blade in the Azure portal.
6. To verify that continuous deployment is occurring from the repository of your choice, push a change to the repository. Your app should update to reflect the changes shortly after the push to the repository completes. You can verify that it has pulled in the update in the **Deployment options** blade of your app.

Continuous deployment of a Visual Studio solution

Pushing a Visual Studio solution to Azure App Service is just as easy as pushing a simple index.html file. The App Service deployment process streamlines all the details, including restoring NuGet dependencies and building the application binaries. You can follow the source control best practices of maintaining code only in your Git repository, and let App Service deployment take care of the rest.

The steps for pushing your Visual Studio solution to App Service are the same as in the [previous section](#), provided that you configure your solution and repository as follows:

- Use the Visual Studio source control option to generate a `.gitignore` file such as the image below or manually add a `.gitignore` file in your repository root with content similar to this [.gitignore sample](#).



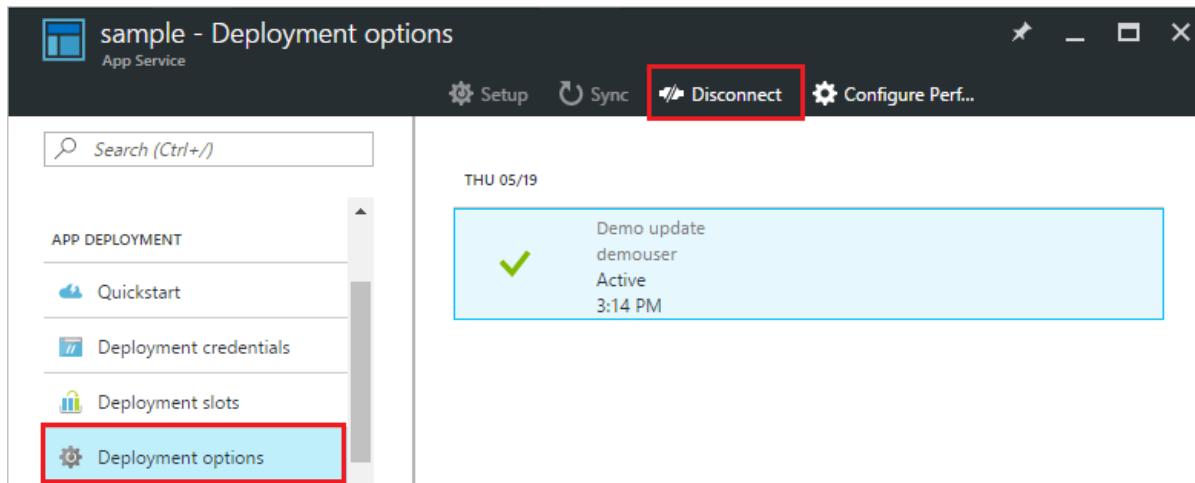
- Add the entire solution's directory tree to your repository, with the .sln file in the repository root.

Once you have set up your repository as described, and configured your app in Azure for continuous publishing from one of the online Git repositories, you can develop your ASP.NET application locally in Visual Studio and continuously deploy your code simply by pushing your changes to your online Git repository.

Disable continuous deployment

To disable continuous deployment,

1. In your app's menu blade in the [Azure portal](#), click **APP DEPLOYMENT > Deployment options**. Then click **Disconnect** in the **Deployment options** blade.



2. After answering **Yes** to the confirmation message, you can return to your app's blade and click **APP DEPLOYMENT > Deployment options** if you would like to set up publishing from another source.

Additional Resources

- [How to investigate common issues with continuous deployment](#)

- [How to use PowerShell for Azure](#)
- [How to use the Azure Command-Line Tools for Mac and Linux](#)
- [Git documentation](#)
- [Project Kudu](#)
- [Use Azure to automatically generate a CI/CD pipeline to deploy an ASP.NET 4 app](#)

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Local Git Deployment to Azure App Service

9/19/2017 • 5 min to read • [Edit Online](#)

This tutorial shows you how to deploy your app to [Azure Web Apps](#) from a Git repository on your local computer.

App Service supports this approach with the **Local Git** deployment option in the [Azure Portal](#).

Many of the Git commands described in this article are performed automatically when creating an App Service app using the [Azure Command-Line Interface](#) as described [here](#).

Prerequisites

To complete this tutorial, you need:

- Git. You can download the installation binary [here](#).
- Basic knowledge of Git.
- A Microsoft Azure account. If you don't have an account, you can [sign up for a free trial](#) or [activate your Visual Studio subscriber benefits](#).

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter app in App Service. No credit cards required; no commitments.

Step 1: Create a local repository

Perform the following tasks to create a new Git repository.

1. Start a command-line tool, such as **GitBash** (Windows) or **Bash** (Unix Shell). On OS X systems you can access the command-line through the **Terminal** application.
2. Navigate to the directory where the content to deploy would be located.
3. Use the following command to initialize a new Git repository:

```
git init
```

Step 2: Commit your content

App Service supports applications created in a variety of programming languages.

1. If your repository already includes content skip this point and move to point 2 below. If your repository does not already include content simply populate with a static .html file as follows:
 - Using a text editor, create a new file named **index.html** at the root of the Git repository
 - Add the following text as the contents for the index.html file and save it: *Hello Git!*
2. From the command-line, verify that you are under the root of your Git repository. Then use the following command to add files to your repository:

```
git add -A
```

1. Next, commit the changes to the repository by using the following command:

```
git commit -m "Hello Azure App Service"
```

Step 3: Enable the App Service app repository

Perform the following steps to enable a Git repository for your App Service app.

1. Log in to the [Azure Portal](#).
2. In your App Service app's blade, click **Settings > Deployment source**. Click **Choose source**, then click **Local Git Repository**, and then click **OK**.

The screenshot shows three overlapping windows:

- Left Window (App Service blade):** Shows the 'Deployment' section with 'Deployment options' selected.
- Middle Window (Deployment source):** Shows 'Choose Source' and 'Local Git Repository' selected.
- Right Window (Choose source):** Shows a list of options: Visual Studio Team Services, OneDrive, Local Git Repository (selected), GitHub, Bitbucket, Dropbox, and External Repository.

3. If this is your first time setting up a repository in Azure, you need to create login credentials for it. You will use them to log into the Azure repository and push changes from your local Git repository. From your app's blade, click **Settings > Deployment credentials**, then configure your deployment username and password. When you're done, click **Save**.

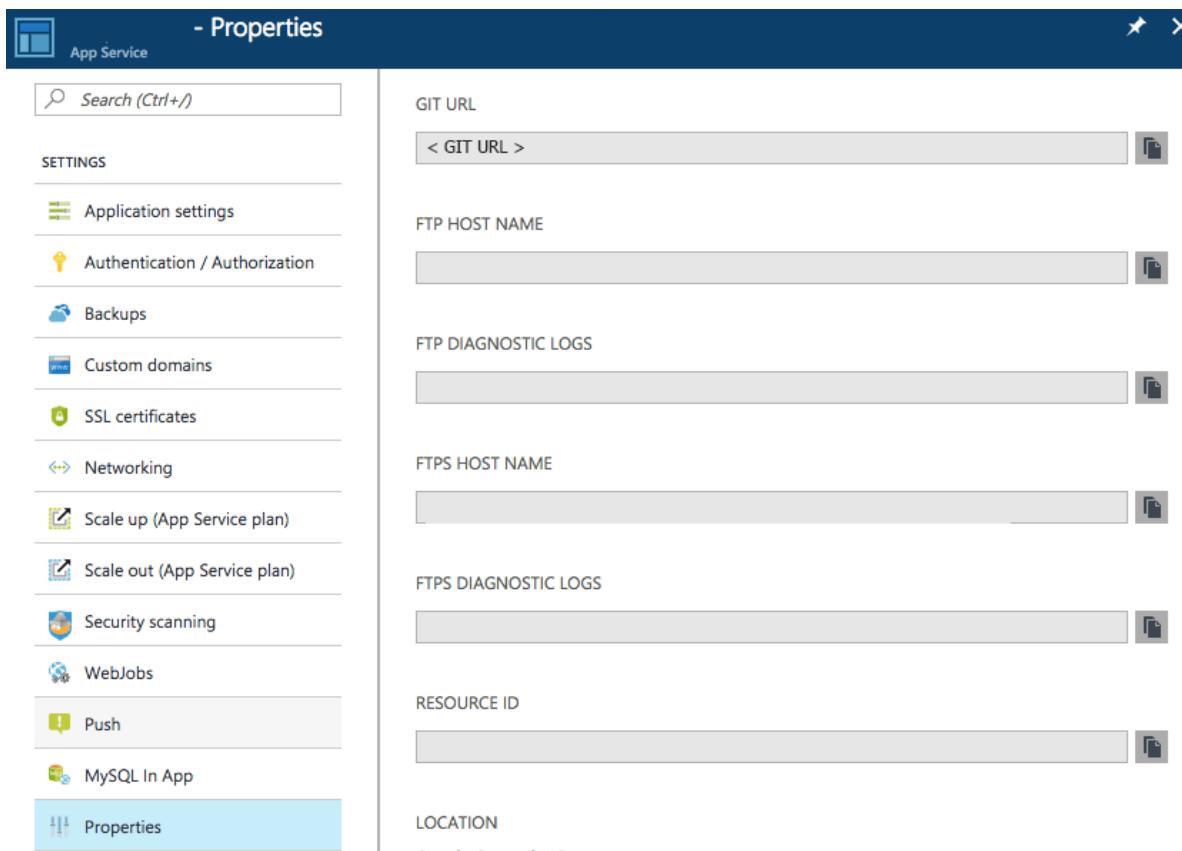
The screenshot shows the 'Deployment credentials' configuration page:

- Left Panel:** Shows the 'Deployment' section with 'Deployment credentials' selected.
- Right Panel:** Shows fields for 'New name and password'. It includes:
 - A note: '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'
 - 'FTP/deployment username': 'your_ftp_username_here' (with a checkmark)
 - 'Password' and 'Confirm password' fields.

Step 4: Deploy your project

Use the following steps to publish your app to App Service using Local Git.

1. In your app's blade in the Azure Portal, click **Settings > Properties** for the **Git URL**.



Git URL is the remote reference to deploy to from your local repository. You'll use this URL in the following steps.

2. Using the command-line, verify that you are in the root of your local Git repository.
3. Use `git remote` to add the remote reference listed in **Git URL** from step 1. Your command will look similar to the following:

```
git remote add azure  
https://<username>@localgitdeployment.scm.azurewebsites.net:443/localgitdeployment.git
```

NOTE

The **remote** command adds a named reference to a remote repository. In this example, it creates a reference named 'azure' for your web app's repository.

4. Push your content to App Service using the new **azure** remote you just created.

```
git push azure master
```

You will be prompted for the password you created earlier when you reset your deployment credentials in the Azure Portal. Enter the password (note that Gitbash does not echo asterisks to the console as you type your password).

1. Go back to your app in the Azure Portal. A log entry of your most recent push should be displayed in the **Deployments** blade.

The screenshot shows the Azure App Service Deployment blade. On the left, a sidebar lists navigation options: DEPLOYMENT (Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview)), SETTINGS (Application settings, Authentication / Authorization, Backups), and a search bar. The main area displays deployment history for 'TUE 03/21'. Three successful deployments are listed:

- updates to tables and forms
<user name>
Active
8:57 AM
- removed footer
<user name>
Inactive
8:25 AM
- Fixed demo page link
<user name>
Inactive
6:28 AM

- Click the **Browse** button at the top of the app's blade to verify the content has been deployed.

Troubleshooting

The following are errors or problems commonly encountered when using Git to publish to an App Service app in Azure:

Symptom: Unable to access '[siteURL]': Failed to connect to [scmAddress]

Cause: This error can occur if the app is not up and running.

Resolution: Start the app in the Azure Portal. Git deployment will not work unless the app is running.

Symptom: Couldn't resolve host 'hostname'

Cause: This error can occur if the address information entered when creating the 'azure' remote was incorrect.

Resolution: Use the `git remote -v` 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.

Symptom: No refs in common and none specified; doing nothing. Perhaps you should specify a branch such as 'master'.

Cause: This error can occur if you do not specify a branch when performing a git push operation, and have not set the push.default value used by Git.

Resolution: Perform the push operation again, specifying the master branch. For example:

```
git push azure master
```

Symptom: src refspec [branchname] does not match any.

Cause: This error can occur if you attempt to push to a branch other than master on the 'azure' remote.

Resolution: Perform the push operation again, specifying the master branch. For example:

```
git push azure master
```

Symptom: RPC failed; result=22, HTTP code = 502.

Cause: This error can occur if you attempt to push a large git repository over HTTPS.

Resolution: Change the git configuration on the local machine to make the postBuffer bigger

```
git config --global http.postBuffer 524288000
```

Symptom: Error - Changes committed to remote repository but your web app not updated.

Cause: This error can occur if you are deploying a Node.js app containing a package.json file that specifies additional required modules.

Resolution: Additional messages containing 'npm ERR!' should be logged prior to this error, and can provide additional context on the failure. The following are known causes of this error and the corresponding 'npm ERR!' message:

- **Malformed package.json file:** npm ERR! Couldn't read dependencies.
- **Native module that does not have a binary distribution for Windows:**
 - npm ERR! `cmd "/c" "node-gyp rebuild"`` failed with 1
OR
 - npm ERR! [modulename@version] preinstall: `make || gmake`

Additional Resources

- [Git documentation](#)
- [Project Kudu documentation](#)
- [Continous Deployment to Azure App Service](#)
- [How to use PowerShell for Azure](#)
- [How to use the Azure Command-Line Interface](#)

Provision and deploy microservices predictably in Azure

9/28/2017 • 14 min 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 apps, API apps, and logic 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.

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

What you will do

In the tutorial, you will deploy an application that includes:

- Two web 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 a web 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](#) like the [Scalable WordPress](#) app can include the MySQL database, storage accounts, the App Service plan, the web app itself, alert rules, app settings, autoscale settings, and more, and all these templates are available to you through PowerShell. For information on how to download and use these templates, see [Using Azure PowerShell with Azure Resource Manager](#).

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 web app, you can extend this to enable deploying an entire resource group by putting an azuredeploy.json file in the repository root. This JSON file, which contains the resource group template, will be used by the Deploy to Azure button to create the resource group. For an example, see the [ToDoApp](#) sample, which you will use in this tutorial.

Get the sample resource group template

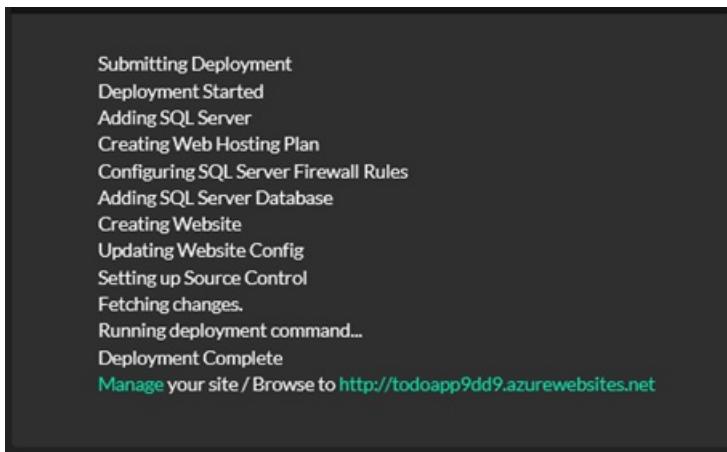
So now let's get right to it.

1. Navigate to the [ToDoApp](#) App Service sample.
2. In readme.md, click **Deploy to Azure**.
3. You're taken to the [deploy-to-azure](#) site and asked to input deployment parameters. Notice that most of the fields are populated with the repository name and some random strings for you. You can change all the fields if you want, but the only things you have to enter are the SQL Server administrative login and the password, then click **Next**.

The screenshot shows the 'Deploy to Azure' configuration interface. It includes the following fields:

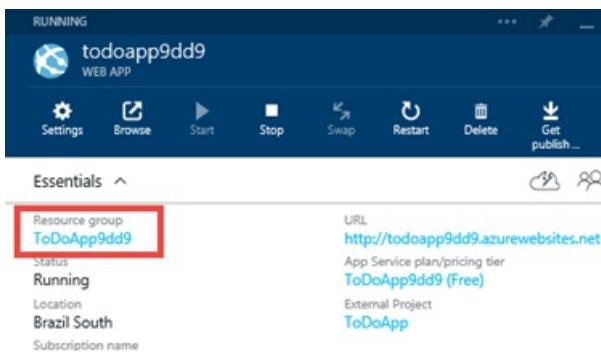
- Git Repository Url: <https://github.com/azure-appservice-samples/ToDoApp>
- Branch: master
- Subscription: Visual Studio Ultimate with MSDN
- Resource Group Name: ToDoApp9dd9
- Site Name: Name is available
- Site Location: Brazil South
- Sku: Free
- Sql Server Name: todoapp9dd9-server
- Sql Server Admin Login: todoapp9dd9-server
- Sql Server Location: East US 2
- Sql Server Admin Password: (empty)
- Sql Db Name: DemosDB
- Sql Db Collation: SQL_Latin1_General_CI_AS
- Sql Db Max Size Bytes: 1073741824
- Sql Db Service Objective Id: 910b4fc8-8a29-4c3e-958f-f7ba794388b2

4. Next, click **Deploy** to start the deployment process. Once the process runs to completion, click the <http://todoappXXXX.azurewebsites.net> link to browse the deployed application.

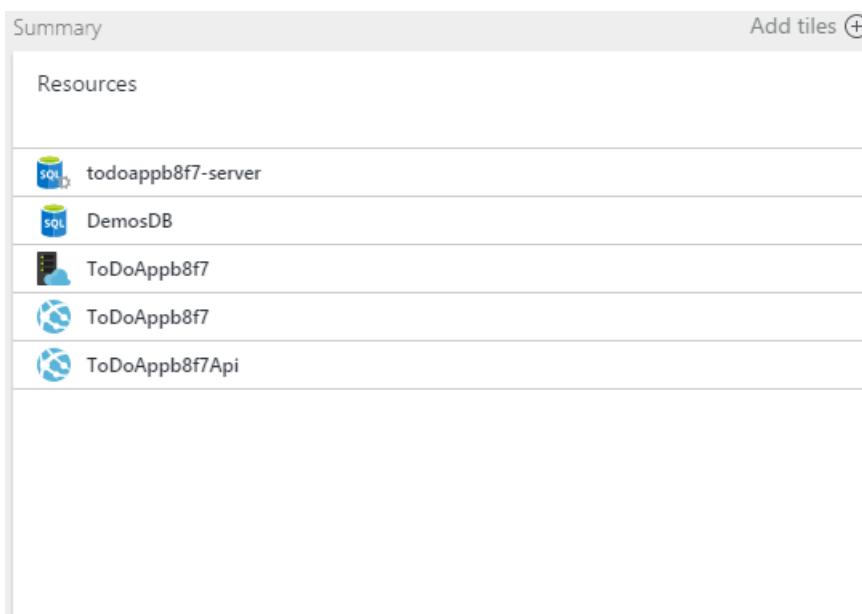


The UI would be a little slow when you first browse to it because the apps are just starting up, but convince yourself that it's a fully-functional application.

5. Back in the Deploy page, click the **Manage** link to see the new application in the Azure Portal.
6. In the **Essentials** dropdown, click the resource group link. Note also that the web app is already connected to the GitHub repository under **External Project**.



7. In the resource group blade, note that there are already two web 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:

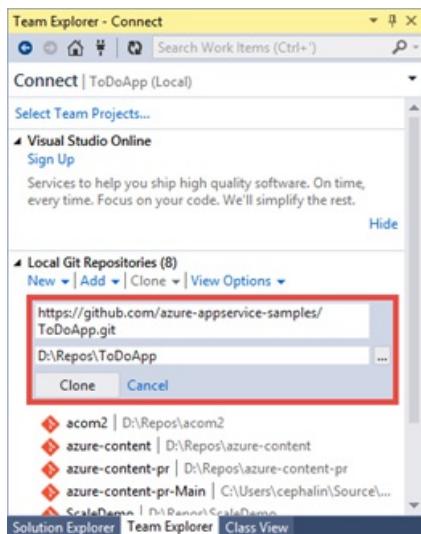
- The Deploy to Azure button
- azuredeploy.json in the repo root

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.

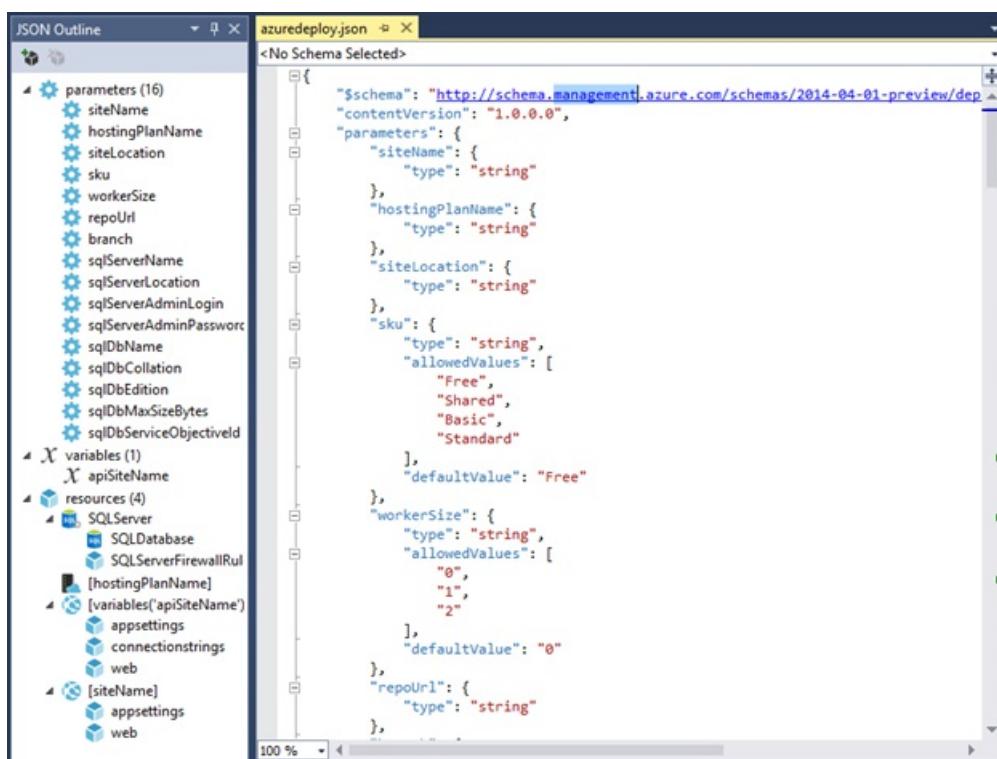
Examine (or edit) AZUREDEPLOY.JSON

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.



2. From the repository root, open azuredeploy.json in Visual Studio. If you don't see the JSON Outline pane, you need to install Azure .NET SDK.



I'm not going to describe every detail of the JSON format, but the [More Resources](#) section has links for learning the resource group template language. Here, I'm just going to show you the interesting features that can help you get started in making your own custom template for app deployment.

Parameters

Take a look at the parameters section to see that most of these parameters are what the **Deploy to Azure** button prompts you to input. The site behind the **Deploy to Azure** button populates the input UI using the parameters defined in `azureddeploy.json`. These parameters are used throughout the resource definitions, such as resource names, property values, etc.

Resources

In the resources node, you can see that 4 top-level resources are defined, including a SQL Server instance, an App Service plan, and two web 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.



```
        "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",
        "location": "[parameters('siteLocation')]",
        "properties": {
            "name": "[variables('apiSiteName')]"
        }
    }
]
```

Note that the `type` element specifies the string for an App Service plan (it was called a server farm a long, long time ago), and other elements and properties are filled in using the parameters defined in the JSON file, and this resource doesn't have any nested resources.

NOTE

Note also that the value of `apiVersion` tells Azure which version of the REST API to use the JSON resource definition with, and it can affect how the resource should be formatted inside the `{}`.

SQL Server

Next, click on the SQL Server resource named **SQLServer** in the JSON Outline.

```

{
  "apiVersion": "2014-04-01-preview",
  "name": "[parameters('sqlServerName')]",
  "type": "Microsoft.Sql/servers",
  "location": "[parameters('sqlServerLocation')]",
  "tags": {
    "displayName": "SQLServer"
  },
  "properties": {
    "administratorLogin": "[parameters('sqlServerAdminLogin')]",
    "administratorLoginPassword": "[parameters('sqlServerAdminPassword')]"
  },
  "resources": [
    {
      "apiVersion": "2014-11-01",
      "name": "[parameters('sqlDbName')]",
      "type": "databases",
      "location": "[parameters('sqlServerLocation')]",
      "tags": {
        "displayName": "SQLDatabase"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
      ],
      "properties": {
        "edition": "[parameters('sqlDbEdition')]",
        "collation": "[parameters('sqlDbCollation')]",
        "maxSizeBytes": "[parameters('sqlDbMaxSizeBytes')]",
        "requestedServiceObjectiveId": "[parameters('sqlDbServiceObjectiveId')]"
      }
    },
    {
      "apiVersion": "2014-11-01",
      "name": "SQLServerFirewallRules",
      "type": "firewallrules",
      "location": "[parameters('sqlServerLocation')]",
      "dependsOn": [
        "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
      ],
      "properties": {
        "endIpAddress": "0.0.0.0",
        "startIpAddress": "0.0.0.0"
      }
    }
  ]
}

```

Note the following about the highlighted JSON code:

- The use of parameters ensures that the created resources are named and configured in a way that makes them consistent with one another.
- The SQLServer resource has two nested resources, each has a different value for `type`.
- The nested resources inside `"resources": [...]`, where the database and the firewall rules are defined, have a `dependsOn` element that specifies the resource ID of the root-level SQLServer resource. This tells Azure Resource Manager, “before you create this resource, that other resource must already exist; and if that other resource is defined in the template, then create that one first”.

NOTE

For detailed information on how to use the `resourceId()` function, see [Azure Resource Manager Template Functions](#).

- The effect of the `dependsOn` element is that Azure Resource Manager can know which resources can be created in parallel and which resources must be created sequentially.

Web app

Now, let's move on to the actual web apps themselves, which are more complicated. Click the `[variables('apiSiteName')]` web 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 web app depends on two different resources. This means that Azure Resource Manager will create the web 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 web 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 web 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 web 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 a web app, you should be able to see web app configuration details similar to the below screenshot:

```

1  {
2    "id": "/subscriptions/62f3ac8c-ca8d-407b-abd8
-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/sites/ToDoApp9dd9Api/config/appsettings",
3    "name": "appsettings",
4    "type": "Microsoft.Web/sites/config",
5    "kind": null,
6    "location": "Brazil South",
7    "tags": {
8      "hidden-related:/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/server-farms/ToDoApp9dd9": "Resource"
9    },
10   "plan": null,
11   "properties": [
12     {
13       "name": "PROJECT",
14       "value": "src\\MultiChannelToDo\\MultiChannelToDo.csproj"
15     },
16     {
17       "name": "clientUrl",
18       "value": "http://ToDoApp9dd9.azurewebsites.net"
19     }
20   ]
21 }

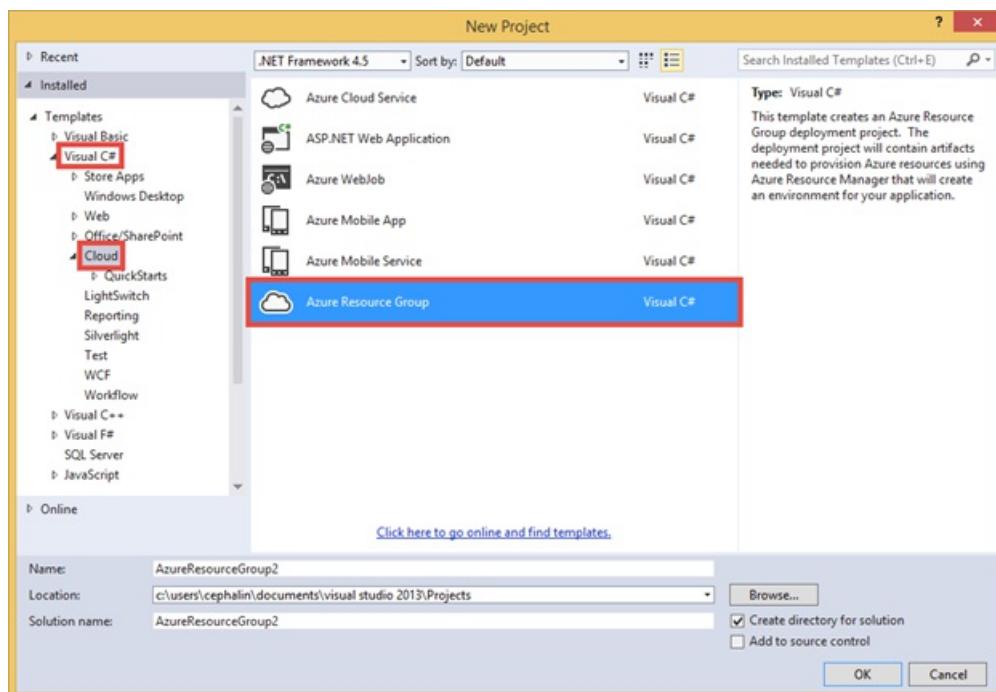
```

Again, the nested resources should have a hierarchy very similar to those in your JSON template file, and you should see the app settings, connection strings, etc., properly reflected in the JSON pane. The absence of settings here may indicate an issue with your JSON file and can help you troubleshoot your JSON template file.

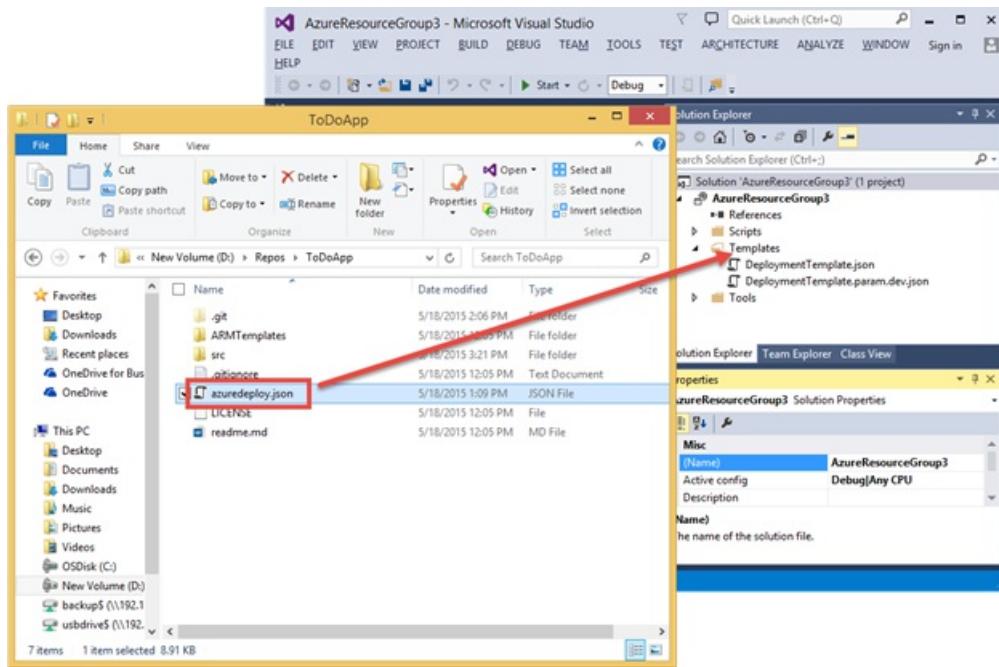
Deploy the resource group template yourself

The **Deploy to Azure** button is great, but it allows you to deploy the resource group template in `azuredeploy.json` only if you have already pushed `azuredeploy.json` to GitHub. The Azure .NET SDK also provides the tools for you to deploy any JSON template file directly from your local machine. To do this, follow the steps below:

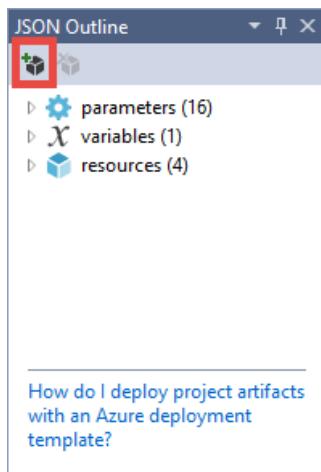
1. In Visual Studio, click **File > New > Project**.
2. Click **Visual C# > Cloud > Azure Resource Group**, then click **OK**.



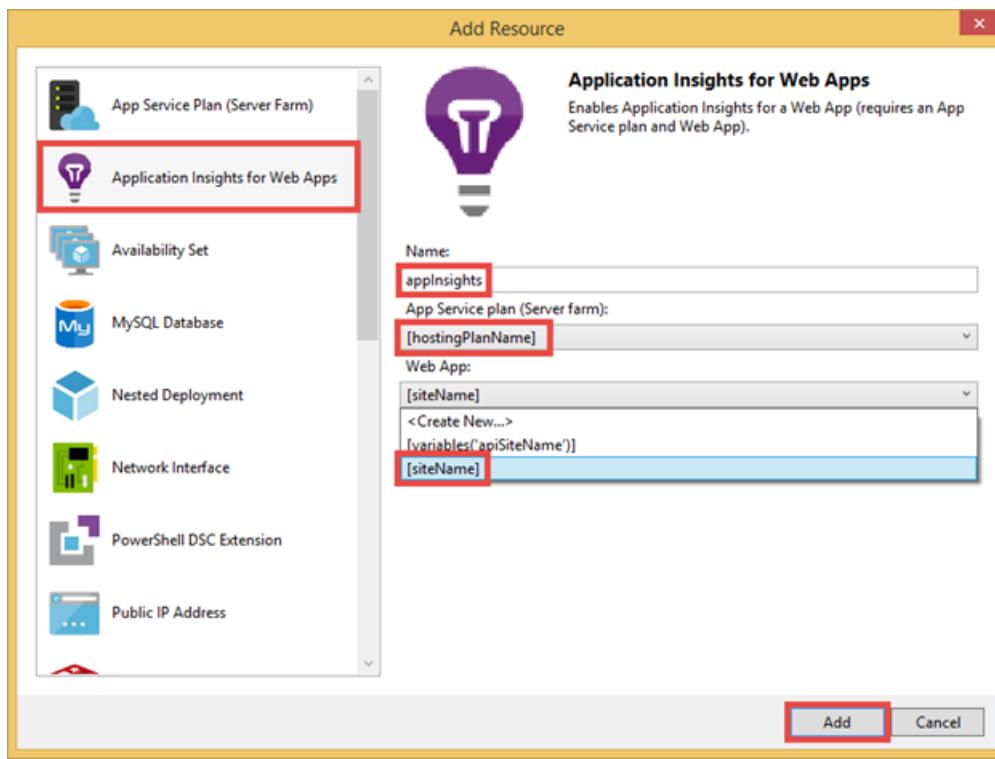
3. In **Select Azure Template**, select **Blank Template** and click **OK**.
4. Drag `azuredeploy.json` into the **Template** folder of your new project.



5. From Solution Explorer, open the copied `azuredelay.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 web 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 web app. These resources are not enabled by their existing definition and you're going to change that.

JSON Outline

- parameters (16)
- variables (1)
- resources (10)
 - SQLServer
 - [hostingPlanName]
 - [variables('apiSiteName')]
 - [siteName]
 - applnights AutoScale
 - CPUHigh applnights
 - LongHttpQueue applnights
 - ServerErrors applnights
 - ForbiddenRequests applnights
 - Component applnights

How do I deploy project artifacts with an Azure deployment template?

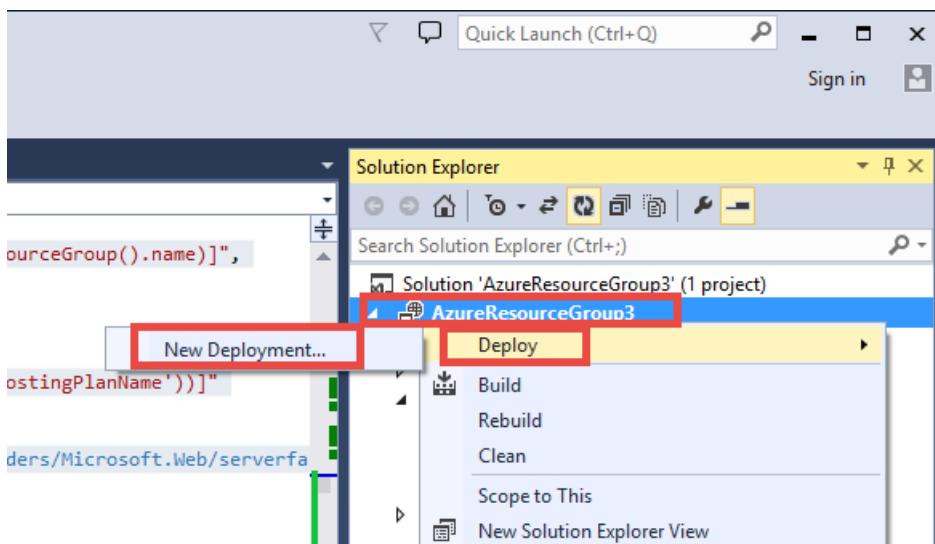
8. In the JSON Outline, click **applnights AutoScale** to highlight its JSON code. This is the scaling setting for your App Service plan.
9. In the highlighted JSON code, locate the `location` and `enabled` properties and set them as shown below.

```
{
  "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
  "type": "Microsoft.Insights/autoscalesettings",
  "location": "[parameters('siteLocation')]",
  "apiVersion": "2014-04-01",
  "dependsOn": [...],
  "tags": [...],
  "properties": {
    "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
    "profiles": [...],
    "enabled": true,
    "targetResourceUri": "[concat(resourceGroup().id, '/providers/Microsoft.W...
  }
},
```

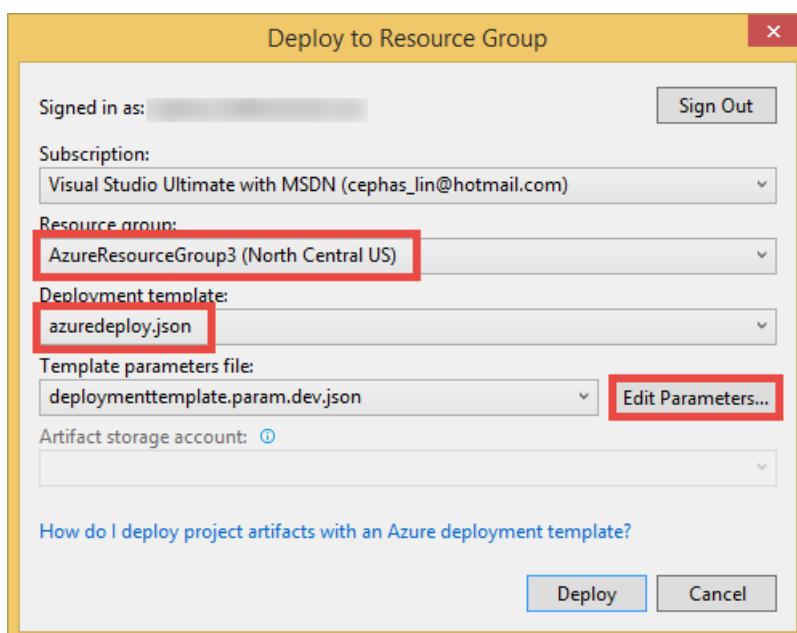
- In the JSON Outline, click **CPUHigh appInsights** to highlight its JSON code. This is an alert.
- Locate the `location` and `isEnabled` properties and set them as shown below. Do the same for the other three alerts (purple bulbs).

```
{
  "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": [...]
  }
},
```

- You're now ready to deploy. Right-click the project and select **Deploy > New Deployment**.

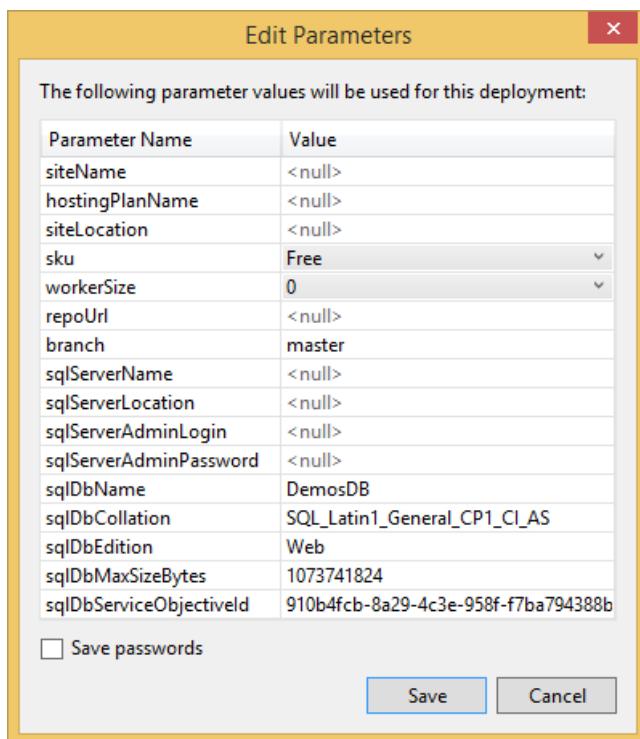


- Log into your Azure account if you haven't already done so.
- Select an existing resource group in your subscription or create a new one, select **azureddeploy.json**, and then click **Edit Parameters**.

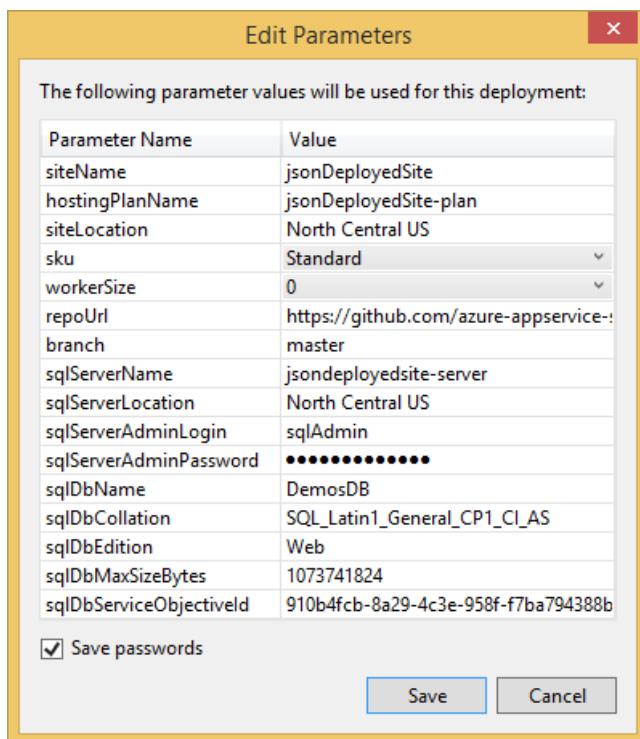


You'll now be able to edit all the parameters defined in the template file in a nice table. Parameters that define defaults will already have their default values, and parameters that define a list of allowed values will

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 "PowerShell", "Az", "1.0.0"
    $headers = [System.Collections.Generic.Dictionary[[String],[String]]]::new()
    $headers.Add("User-Agent", $AzureToolsUserAgentString)
    $client = [System.Net.Http.HttpClient]::new()
    $client.DefaultRequestHeaders = $headers
}

# Create or update the resource group using the specified template file and template parameters file
Switch-AzureMode AzureResourceManager
New-AzureResourceGroup -Name $ResourceGroupName `
    -Location $ResourceGroupLocation `
    -TemplateFile $Templatefile `
    -TemplateParameterFile $TemplateParametersFile `
    @OptionalParameters `
    -Force -Verbose
```

The last cmdlet, `New-AzureResourceGroup`, is the one that actually performs the action. All this should demonstrate to you that, with the help of tooling, it is relatively straightforward to deploy your cloud application predictably. Every time you run the cmdlet on the same template with the same parameter file, you're going to get the same result.

Summary

In DevOps, repeatability and predictability are keys to any successful deployment of a high-scale application composed of microservices. In this tutorial, you have deployed a two-microservice application to Azure as a single resource group using the Azure Resource Manager template. Hopefully, it has given you the knowledge you need in order to start converting your application in Azure into a template and can provision and deploy it predictably.

More resources

- [Azure Resource Manager Template Language](#)
- [Authoring Azure Resource Manager Templates](#)
- [Azure Resource Manager Template Functions](#)
- [Deploy an application with Azure Resource Manager template](#)
- [Using Azure PowerShell with Azure Resource Manager](#)
- [Troubleshooting Resource Group Deployments in Azure](#)

Configure deployment credentials for Azure App Service

12/14/2017 • 2 min to read • [Edit Online](#)

Azure App Service supports two types of credentials for [local Git deployment](#) and [FTP/S deployment](#). These are not the same as your Azure Active Directory 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. These are the default credentials set that you configure in **App Services > <app_name> > Deployment credentials**. This is also the default set that's surfaced in the portal GUI (such as the **Overview** and **Properties** of your app's [resource page](#)).

NOTE

When you delegate access to Azure resources via Role Based Access Control (RBAC) or co-admin permissions, each Azure user that receives access to an app can use his/her personal user-level credentials until access is revoked. These deployment credentials should not be shared 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 is generated automatically at app creation, and is found in the app's publish profile. You cannot manually configure the credentials, but you can reset them for an app anytime.

NOTE

In order to give someone access to these credentials via Role Based Access Control (RBAC), you need to make them contributor or higher on the Web App. Readers are not allowed to publish, and hence can't access those credentials.

Set and reset 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.

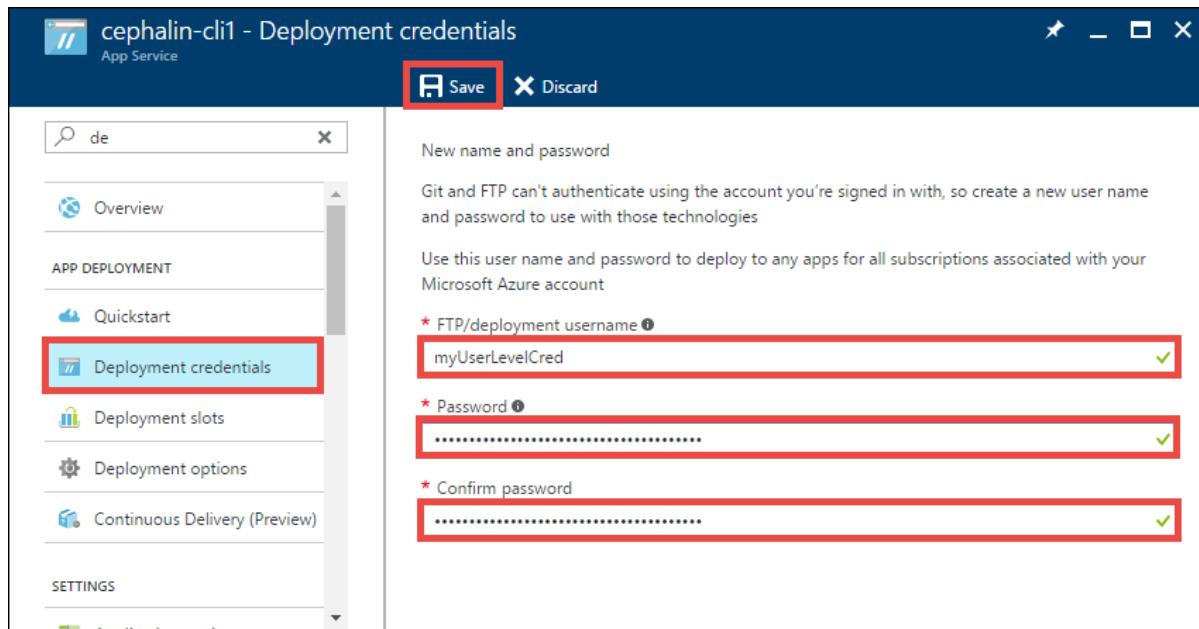
To configure your user-level credentials:

1. In the [Azure portal](#), click App Service > **<any_app>** > **Deployment credentials**.

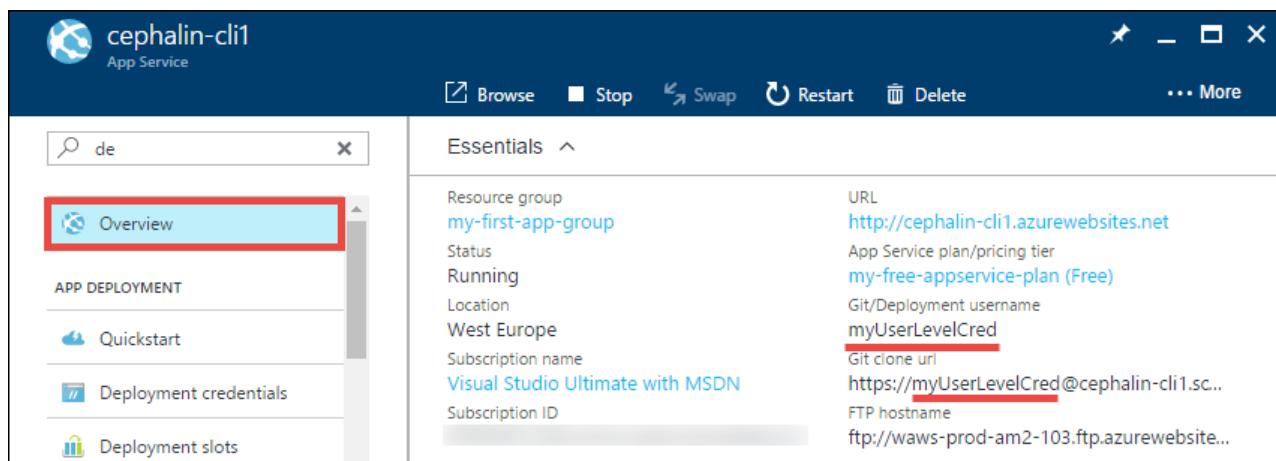
NOTE

In the portal, you must have at least one app before you can access the deployment credentials page. However, with the [Azure CLI](#), you can configure user-level credentials without an existing app.

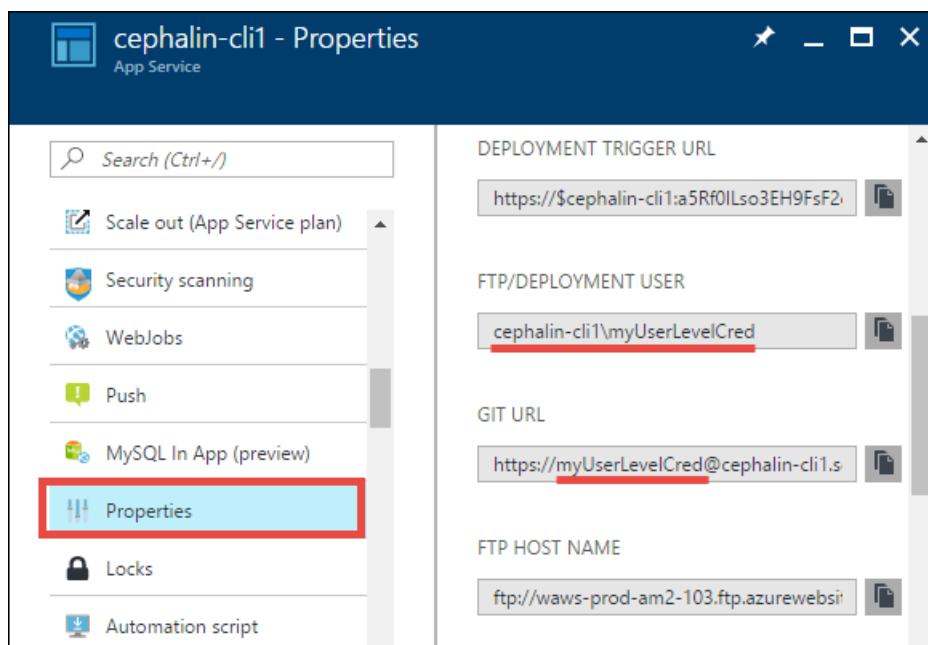
2. Configure the user name and password, and then click **Save**.



Once you have set your deployment credentials, you can find the *Git* deployment username in your app's **Overview**,



and and *FTP* deployment username in your app's **Properties**.



NOTE

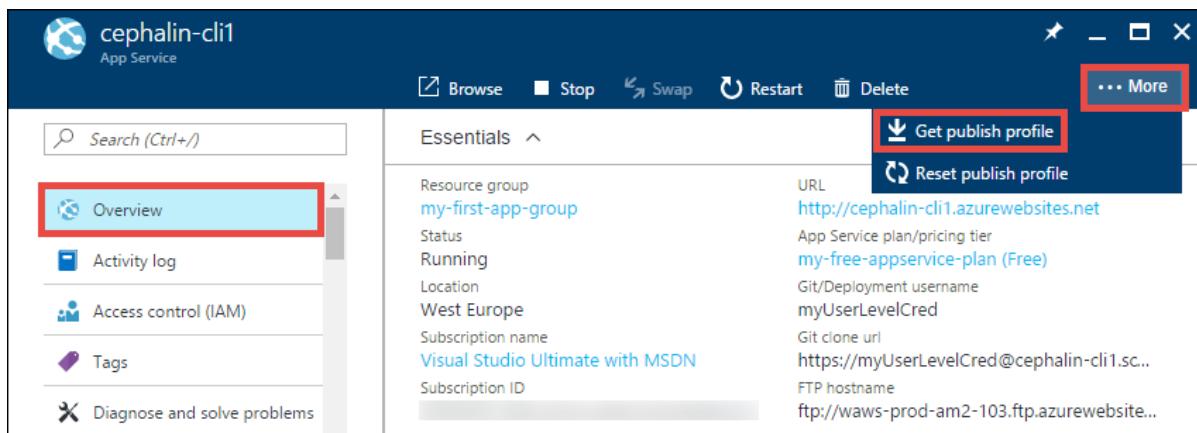
Azure does not show your user-level deployment password. If you forget the password, you can't retrieve it. However, you can reset your credentials by following the steps in this section.

Get and reset app-level credentials

For each app in App Service, its app-level credentials are stored in the XML publish profile.

To get the app-level credentials:

1. In the [Azure portal](#), click App Service > **<any_app>** > **Overview**.
2. Click **...More** > **Get publish profile**, and download starts for a .PublishSettings file.



3. Open the .PublishSettings file and find the `<publishProfile>` tag with the attribute `publishMethod="FTP"`. Then, get its `userName` and `password` attributes. These are the app-level credentials.

```
<publishData>
  <publishProfile profileName="cephalin-cli1 - Web Deploy"
    publishMethod="MSDeploy"
    publishUrl="cephalin-cli1.scm.azurewebsites.net:443"
    msdeploySite="cephalin-cli1"
    userName="$cephalin-cli1"
    userPWD="a5Rf0lLso3EH9FsF2cbgfbqGHsriY4x4cWE3lbaNi3aqGwnzr9sZy4v2C3E9"
    destinationAppUrl="http://cephalin-cli1.azurewebsites.net"
    SQLServerDBConnectionString=""
    mySQLDBConnectionString=""
    hostingProviderForumLink=""
    controlPanellink=""
    webSystem="WebSites">
    <databases />
  </publishProfile>
  <publishProfile profileName="cephalin-cli1 - FTP"
    publishMethod="FTP"
    publishUrl="ftp://waws-prod-am2-103.ftp.azurewebsites.windows.net/site/wwwroot"
    ftpPassiveMode="True"
    userName="cephalin-cli1\$cephalin-cli1"
    userPWD="a5Rf0lLso3EH9FsF2cbgfbqGHsriY4x4cWE3lbaNi3aqGwnzr9sZy4v2C3E9"
    destinationAppUrl="http://cephalin-cli1.azurewebsites.net"
    SQLServerDBConnectionString=""
    mySQLDBConnectionString=""
    hostingProviderForumLink=""
    controlPanellink=""
    webSystem="WebSites">
    <databases />
  </publishProfile>
</publishData>
```

Similar to the user-level credentials, the FTP deployment username is in the format of

`<app_name>\<username>`, and the Git deployment username is just `<username>` without the preceding `<app_name>\`.

To reset the app-level credentials:

1. In the [Azure portal](#), click App Service > **<any_app> > Overview**.
2. Click ...**More** > **Reset publish profile**. Click **Yes** to confirm the reset.

The reset action invalidates any previously-downloaded .PublishSettings files.

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

9/19/2017 • 10 min to read • [Edit Online](#)

When you deploy your web app, web app on Linux, mobile back end, and API app to [App Service](#), you can deploy to a separate deployment slot instead of the default production slot when running in the **Standard or Premium** App Service plan mode. Deployment slots are actually live apps with their own hostnames. App content and configurations elements can be swapped between two deployment slots, including the production slot. Deploying your application to a deployment 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 ensures 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 as a result of swap operations. This entire workflow can be automated by configuring [Auto Swap](#) when pre-swap validation is not needed.
- After a swap, the slot with previously staged app now has the previous production app. If the changes swapped into the production slot are not as you expected, you can perform the same swap immediately to get your "last known good site" back.

Each App Service plan mode supports a different number of deployment slots. To find out the number of slots your app's mode supports, see [App Service Pricing](#).

- When your app has multiple slots, you cannot change the mode.
- Scaling is not available for non-production slots.
- Linked resource management is not supported for non-production slots. In the [Azure Portal](#) only, you can avoid this potential impact on a production slot by temporarily moving the non-production slot to a different App Service plan mode. Note that the non-production slot must once again share the same mode with the production slot before you can swap the two slots.

Add a deployment slot

The app must be running in the **Standard or Premium** mode in order for you to enable multiple deployment slots.

1. In the [Azure Portal](#), open your app's [resource blade](#).
2. Choose the **Deployment slots** option, then click **Add Slot**.

The screenshot shows the Azure portal interface for an App Service named 'mywordpresswebapp1'. The main area is titled 'Deployment slots' and contains a message: 'You haven't added any deployment slots. Click ADD SLOT to get started.' At the top right, there are window control buttons (minimize, maximize, close). Below the title, there's a search bar labeled 'Search (Ctrl+ /)'. On the left, a sidebar lists several options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, APP DEPLOYMENT (Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview)). The 'Deployment slots' option is highlighted with a red box. At the top center, there's a red box around the '+ Add Slot' button.

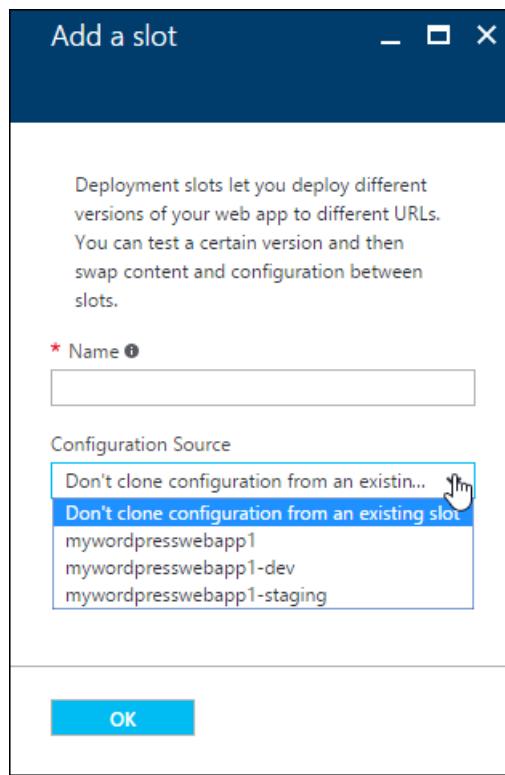
NOTE

If the app is not already in the **Standard** or **Premium** mode, you will receive a message indicating the supported modes for enabling staged publishing. At this point, you have the option to select **Upgrade** and navigate to the **Scale** tab of your app before continuing.

3. In the **Add a slot** blade, give the slot a name, and select whether to clone app configuration from another existing deployment slot. Click the check mark to continue.

The screenshot shows the 'Add a slot' blade. The title is 'Add a slot'. The main content area has a descriptive paragraph about deployment slots. Below it, there are two input fields: 'Name' (with a red box around it) containing 'staging' and a green checkmark icon, and 'Configuration Source' (with a red box around it) containing 'Don't clone configuration from an existin...'. At the bottom, there's a large blue 'OK' button with a red box around it.

The first time you add a slot, you will only have two choices: clone configuration from the default slot in production or not at all. After you have created several slots, you will be able to clone configuration from a slot other than the one in production:



4. In your app's resource blade, click **Deployment slots**, then click a deployment slot to open that slot's resource blade, with a set of metrics and configuration just like any other app. The name of the slot is shown at the top of the blade to remind you that you are viewing the deployment slot.

NAME	STATUS	APP SERVICE PLAN
mywordpresswebapp1-staging	Running	Default1
mywordpresswebapp1-dev	Running	Default1

5. Click the app URL in the slot's blade. Notice the deployment slot has its own hostname and is also a live app. To limit public access to the deployment slot, see [App Service Web App – block web access to non-production deployment slots](#).

There is no content after deployment slot creation. You can deploy to the slot from a different repository branch, or an altogether different repository. You can also change the slot's configuration. Use the publish profile or deployment credentials associated with the deployment slot for content updates. For example, you can [publish to this slot with git](#).

Configuration for deployment slots

When you clone configuration from another deployment slot, the cloned configuration is editable. Furthermore, some configuration elements will follow the content across a swap (not slot specific) while other configuration elements will stay in the same slot after a swap (slot specific). The following lists show the configuration that will 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
- Monitoring and diagnostic settings
- WebJobs content

Settings that are not swapped:

- Publishing endpoints
- Custom Domain Names
- SSL certificates and bindings
- Scale settings
- WebJobs schedulers

To configure an app setting or connection string to stick to a slot (not swapped), access the **Application Settings** blade for a specific slot, then select the **Slot Setting** box for the configuration elements that should stick the slot. Note that marking a configuration element as slot specific has the effect of establishing that element as not swappable across all the deployment slots associated with the app.

App settings

WEBSITE_NODE ✓	0.10.32	<input type="checkbox"/> Slot setting
SLOT_SETTING ✓	TEST	<input checked="" type="checkbox"/> Slot setting
Key	Value	<input type="checkbox"/> Slot setting

Swap deployment slots

You can swap deployment slots in the **Overview** or **Deployment slots** view of your app's resource blade.

IMPORTANT

Before you swap an app from a deployment slot into production, make sure that all non-slot specific settings are configured exactly as you want to have it in the swap target.

1. To swap deployment slots, click the **Swap** button in the command bar of the app or in the command bar of a deployment slot.

The screenshot shows the Azure portal interface for a 'staging Web App'. The top navigation bar includes 'Browse', 'Stop', 'Swap' (which is highlighted with a red box), 'Restart', 'Delete', and 'More'. On the left, a sidebar lists 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'APP DEPLOYMENT' (with 'Quickstart' and 'Deployment credentials' listed under it). The main content area is titled 'Requests and errors' and displays a chart from 12:15 PM to 1 PM. The chart shows values of 100, 80, 60, 40, and 0, with a dark grey bar at the bottom labeled 'No available data.'

2. Make sure that the swap source and swap target are set properly. Usually, the swap target is the production slot. Click **OK** to complete the operation. When the operation finishes, the deployment slots have been swapped.

The screenshot shows the 'Swap' dialog box. The 'Swap type' dropdown is set to 'Swap'. The 'Source' dropdown is set to 'staging' and is highlighted with a red box. The 'Destination' dropdown is set to 'production' and is also highlighted with a red box. Below these fields, there is a 'Preview Changes' section with a lock icon and the text 'No Warnings'. At the bottom of the dialog is a large blue 'OK' button, which is also highlighted with a red box.

For the **Swap with preview** swap type, see [Swap with preview \(multi-phase swap\)](#).

Swap with preview (multi-phase swap)

Swap with preview, or multi-phase swap, simplify validation of slot-specific configuration elements, such as connection strings. For mission-critical workloads, you want to validate that the app behaves as expected when the production slot's configuration is applied, and you must perform such validation *before* the app is swapped into production. Swap with preview is what you need.

NOTE

Swap with preview is not supported in web apps on Linux.

When you use the **Swap with preview** option (see [Swap deployment slots](#)), App Service does the following:

- Keeps the destination slot unchanged so existing workload on that slot (e.g. production) is not impacted.
- Applies the configuration elements of the destination slot to the source slot, including the slot-specific connection strings and app settings.
- Restarts the worker processes on the source slot using these aforementioned configuration elements.
- When you complete the swap: Moves the pre-warmed-up source slot into the destination slot. The destination slot is moved into the source slot as in a manual swap.
- When you cancel the swap: Reapplies the configuration elements of the source slot to the source slot.

You can preview exactly how the app will behave with the destination slot's configuration. Once you complete validation, you complete the swap in a separate step. This step has the added advantage that the source slot is already warmed up with the desired configuration, and clients will not experience any downtime.

Samples for the Azure PowerShell cmdlets available for multi-phase swap are included in the [Azure PowerShell cmdlets for deployment slots](#) section.

Configure Auto Swap

Auto Swap streamlines DevOps scenarios where you want to continuously deploy your app with zero cold start and zero downtime for end customers of the app. When a deployment slot is configured for Auto Swap into production, every time you push your code update to that slot, App Service will automatically swap the app into production after it has already warmed up in the slot.

IMPORTANT

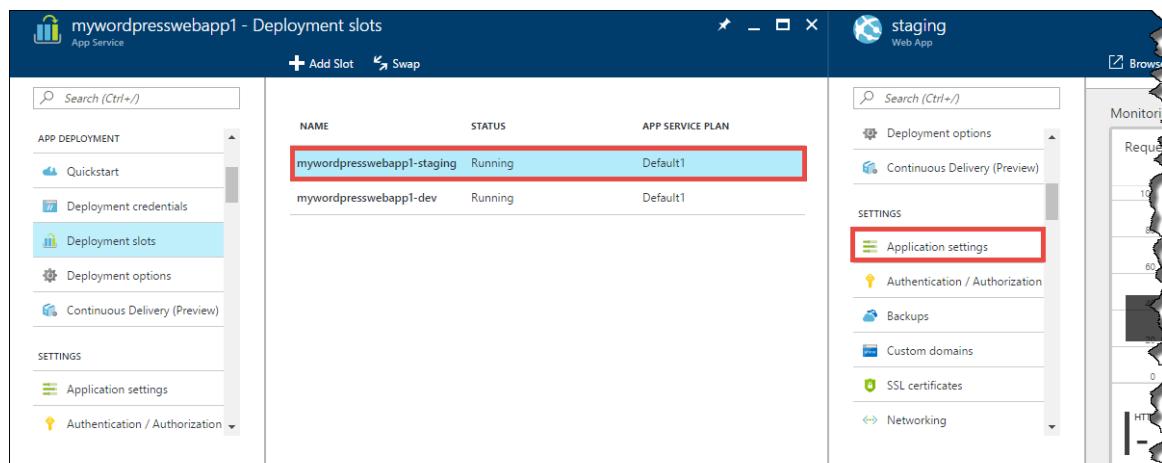
When you enable Auto Swap for a slot, make sure the slot configuration is exactly the configuration intended for the target slot (usually the production slot).

NOTE

Auto swap is not supported in web apps on Linux.

Configuring Auto Swap for a slot is easy. Follow the steps below:

1. In **Deployment Slots**, select a non-production slot, and choose **Application Settings** in that slot's resource blade.



2. Select **On** for **Auto Swap**, select the desired target slot in **Auto Swap Slot**, and click **Save** in the command bar. Make sure configuration for the slot is exactly the configuration intended for the target slot.

The **Notifications** tab will flash a green **SUCCESS** once the operation is complete.

The screenshot shows the 'staging - Application settings' blade for a 'Web App'. The left sidebar lists 'Deployment options', 'Continuous Delivery (Preview)', and 'SETTINGS' with 'Application settings' selected. The main area has a red box around the 'Save' button. A tooltip message says: 'You can improve the performance of your state-less applications by turning off the Affinity Cookie, state-full applications should keep the Affinity Cookie turned on for increased compatibility. Click to learn more.' Below it, 'ARR Affinity' is set to 'Off' (button is grey). Under 'Auto swap destinations cannot be configured from production slot', 'Auto Swap' is set to 'On' (button is purple) and 'Auto Swap Slot' is set to 'production' (button is red). Under 'Debugging', 'Remote debugging' is set to 'Off' (button is grey). At the bottom, 'Remote Visual Studio version' has three options: 2012 (selected), 2013, and 2015.

NOTE

To test Auto Swap for your app, you can first select a non-production target slot in **Auto Swap Slot** to become familiar with the feature.

3. Execute a code push to that deployment slot. Auto Swap will happen after a short time and the update will be reflected at your target slot's URL.

To rollback a production app after swap

If any errors are identified in production after a slot swap, roll the slots back to their pre-swap states by swapping the same two slots immediately.

Custom warm-up before swap

Some apps may require custom warm-up actions. The `<applicationInitialization>` configuration element in `web.config` allows you to specify custom initialization actions to be performed before a request is received. The swap operation will wait for this custom warm-up to complete. Here is a sample `web.config` fragment.

```
<applicationInitialization>
  <add initializationPage="/" hostName="[app hostname]" />
  <add initializationPage="/Home/About" hostname="[app hostname]" />
</applicationInitialization>
```

To delete a deployment slot

In the blade for a deployment slot, open the deployment slot's blade, click **Overview** (the default page), and click **Delete** in the command bar.

The screenshot shows the Azure portal interface for a 'staging' Web App. At the top, there are navigation icons for 'Browse', 'Stop', 'Swap', 'Restart', and 'Delete'. The 'Delete' button is highlighted with a red box. To the right, there's a link to '... More'. Below the top bar, there's a search bar labeled 'Search (Ctrl+ /)'. On the left, a sidebar menu includes 'Overview' (highlighted with a red box), 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. Under 'APP DEPLOYMENT', there are links for 'Quickstart', 'Deployment credentials', and 'Deployment slots'. The main content area is titled 'Monitoring' and contains a chart titled 'Requests and errors'. The chart shows a single data series with values from 0 to 100. A dark grey bar at the bottom of the chart area displays the text 'No available data.'.

Azure PowerShell cmdlets for deployment slots

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-AzureRmWebApp -ResourceGroupName [resource group name] -Name [app name] -Location [location] -AppServicePlan [app service plan name]
```

Create a deployment slot

```
New-AzureRmWebAppSlot -ResourceGroupName [resource group name] -Name [app name] -Slot [deployment slot name] -AppServicePlan [app service plan name]
```

Initiate a swap with review (multi-phase swap) and apply destination slot configuration to source slot

```
$ParametersObject = @{"targetSlot = "[slot name - e.g. "production"]"}  
Invoke-AzureRmResourceAction -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 source slot configuration

```
Invoke-AzureRmResourceAction -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-AzureRmResourceAction -ResourceGroupName [resource group name] -ResourceType Microsoft.Web/sites/slots
-ResourceName [app name]/[slot name] -Action slotsswap -Parameters $ParametersObject -ApiVersion 2015-07-01
```

Delete deployment slot

```
Remove-AzureRmResource -ResourceGroupName [resource group name] -ResourceType Microsoft.Web/sites/slots -Name
[app name]/[slot name] -ApiVersion 2015-07-01
```

Azure Command-Line Interface (Azure CLI) commands for Deployment Slots

The Azure CLI provides cross-platform commands for working with Azure, including support for managing App Service deployment slots.

- For instructions on installing and configuring the Azure CLI, including information on how to connect Azure CLI to your Azure subscription, see [Install and Configure the Azure CLI](#).
- To list the commands available for Azure App Service in the Azure CLI, call `azure site -h`.

NOTE

For Azure CLI 2.0 commands for deployment slots, see [az appservice web deployment slot](#).

azure site list

For information about the apps in the current subscription, call **azure site list**, as in the following example.

```
azure site list webappslotstest
```

azure site create

To create a deployment slot, call **azure site create** and specify the name of an existing app and the name of the slot to create, as in the following example.

```
azure site create webappslotstest --slot staging
```

To enable source control for the new slot, use the **--git** option, as in the following example.

```
azure site create --git webappslotstest --slot staging
```

azure site swap

To make the updated deployment slot the production app, use the **azure site swap** command to perform a swap operation, as in the following example. The production app will not experience any down time, nor will it undergo a cold start.

```
azure site swap webappslotstest
```

azure site delete

To delete a deployment slot that is no longer needed, use the **azure site delete** command, as in the following example.

```
azure site delete webappslotstest --slot staging
```

NOTE

See a web app in action. [Try App Service](#) immediately and create a short-lived starter app—no credit card required, no commitments.

Next Steps

[Azure App Service Web App – block web access to non-production deployment slots](#) [Introduction to App Service on Linux](#) [Microsoft Azure Free Trial](#)

Buy a custom domain name for Azure Web Apps

1/2/2018 • 8 min to read • [Edit Online](#)

App Service domains (preview) are top-level domains that are managed directly in Azure. They make it easy to manage custom domains for [Azure Web Apps](#). This tutorial shows you how to buy an App Service domain and assign DNS names to Azure Web Apps.

This article is for Azure App Service (Web Apps, API Apps, Mobile Apps, Logic Apps). 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 VM as other App Service apps. Some apps may belong to other customers. These tiers are intended to be used only for development and testing purposes.

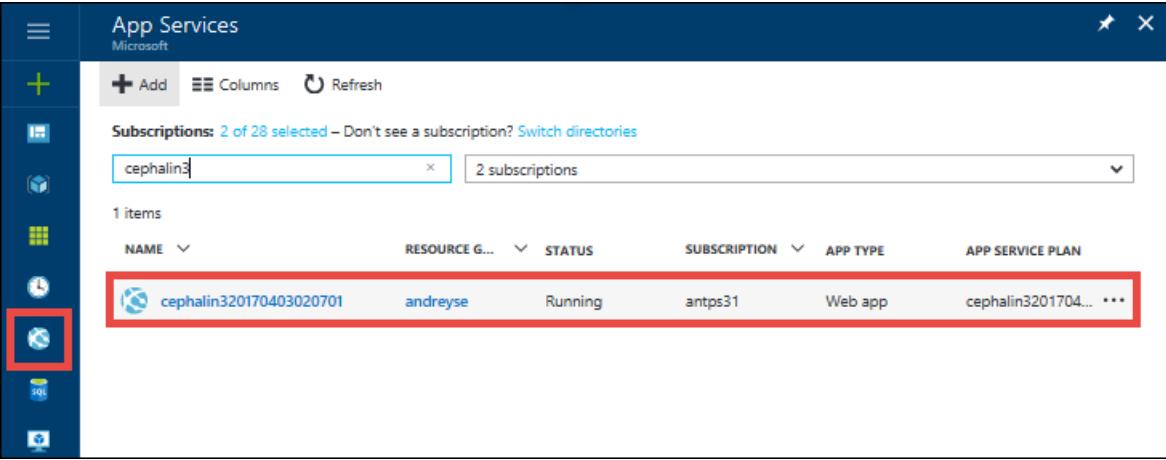
To use custom domains in Azure Web Apps, your web app's [App Service plan](#) must be a paid tier (**Shared**, **Basic**, **Standard**, or **Premium**). In this step, you make sure that the web app is in the supported pricing tier.

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.

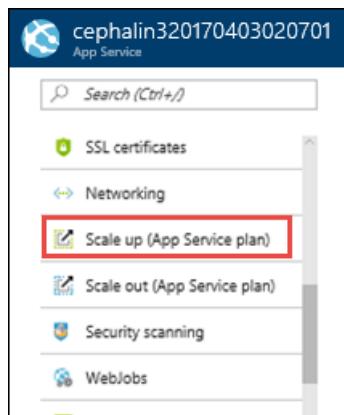


NAME	RESOURCE G...	STATUS	SUBSCRIPTION	APP TYPE	APP SERVICE PLAN
cephalin320170403020701	andreyse	Running	antps31	Web app	cephalin3201704...

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)**.



The app's current tier is highlighted by a blue border. Check to make sure that the app is not in the **Free** tier. Custom DNS is not supported in the **Free** tier.

USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)
F1 Free	D1 Shared*
- Shared infrastructure	- Shared infrastructure
1 GB Storage	1 GB Storage
0.00	9.67
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED, *PER AP...)

Select

If the App Service plan is not **Free**, close the **Choose your pricing tier** page and skip to [Buy the domain](#).

Scale up the App Service plan

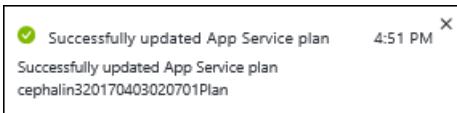
Select any of the non-free tiers (**Shared**, **Basic**, **Standard**, or **Premium**).

Click **Select**.

Choose your pricing tier		
Browse the available plans and their features		
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)
F1 Free	D1 Shared*	
- Shared infrastructure	- Shared infrastructure	
1 GB Storage	1 GB Storage	
0.00	9.67	
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED, *PER AP...)	

Select

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



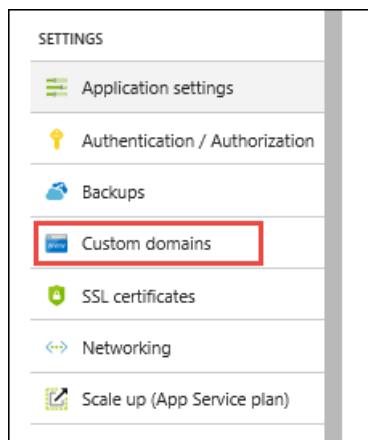
Buy the domain

Sign in to Azure

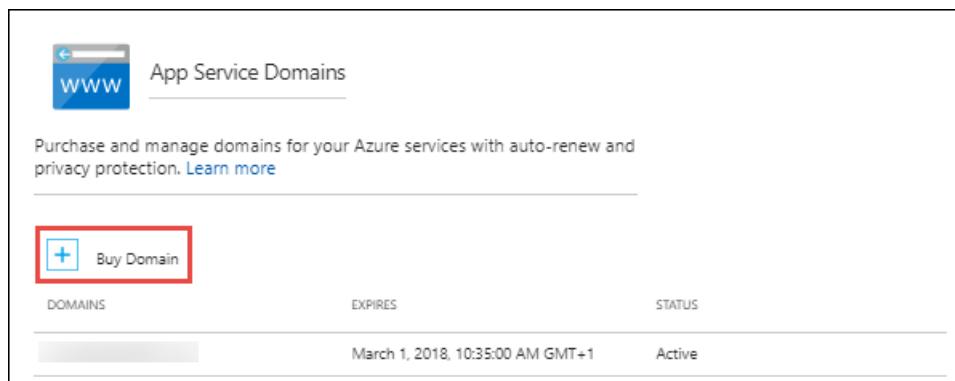
Open the [Azure portal](#) and sign in with your Azure account.

Launch Buy domains

In the **Web Apps** tab, click the name of your web app, select **Settings**, and then select **Custom domains**



In the **Custom domains** page, click **Buy Domain**.

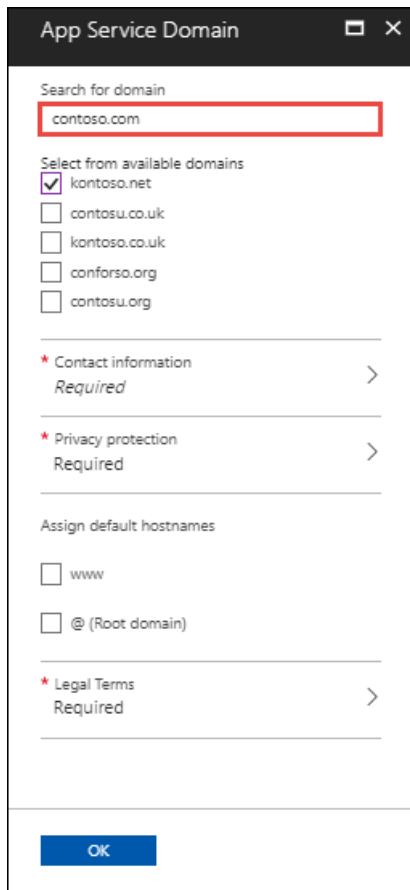


NOTE

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 web app can be accessed at the selected hostnames. If the web 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 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 web 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.

A screenshot of the 'Custom Hostnames' section in the Azure portal. It shows configuration options like IP address (13.95.93.152) and HTTPS Only (set to Off). Below these, a list of hostnames assigned to the site is shown, with 'kontoso.net' and 'www.kontoso.net' highlighted by a red box.

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 web app

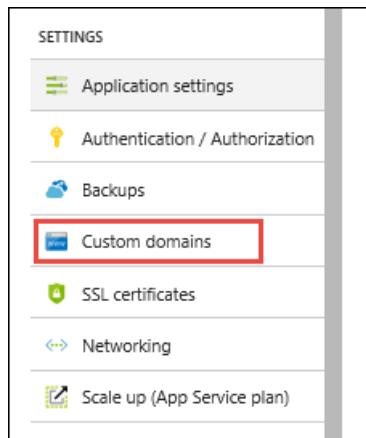
If you choose not to assign one or more default hostnames to your web app during the purchase process, or if you need to assign a hostname not listed, you can assign a hostname at anytime.

You can also assign hostnames in the App Service Domain to any other web app. The steps depend on whether the App Service Domain and the web app belong to the same subscription.

- Different subscription: Map custom DNS records from the App Service Domain to the web 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 a web app, see [Map an existing custom DNS name to Azure Web Apps](#).
- Same subscription: Use the following steps.

Launch add hostname

In the **App Services** page, select the name of your web 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		
Buy Domain		
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 web app's **Custom domains** page. If your domain is in the web app's subscription, but you cannot see it in the web 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

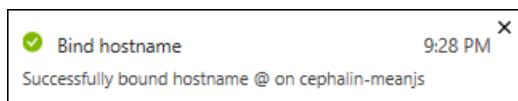
In the **Add hostname** dialog, type the fully qualified domain name of your App Service Domain or any subdomain. For example:

- 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 web 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	...

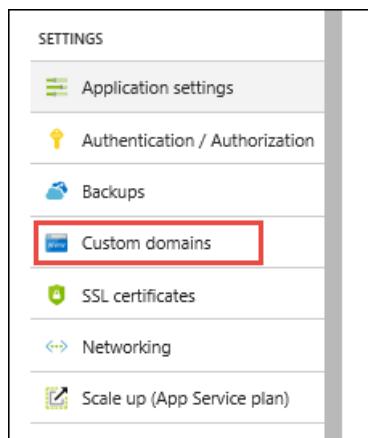
Test the hostnames

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. If you want to turn off automatic renewal, or if you want to manually renew your domain, follow the steps here.

In the **Web Apps** tab, click the name of your web app, select **Settings**, and then select **Custom domains**.



In the **App Service Domains** section, select the domain you want to configure.

App Service Domains		
Buy Domain		
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**.

kontoso.net - Domain renewal

App Service Domain

Search (Ctrl+ /)

Overview

Access control (IAM)

Tags

SETTINGS

Properties

Locks

Automation script

DOMAIN MANAGEMENT

Hostname bindings

Domain renewal

DNS zone

Domain renewal

App Service domains can be set to auto renew to prevent expiration and un-expected domain ownership loss.

Auto renew domain:

Manual domain renewal can only be performed up to 90 days ahead of domain expiration and up to 18 days after domain expiration. You can enable domain auto-renewal policy to reduce the management overhead of this operations.

Expiration date: November 27, 2018 9:04:20 AM CET

Renew domain

To manually renew your domain, select **Renew domain**. However, this button is not active until 90 days before the domain's expiration.

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 **More Services > App Service Domains**.

Microsoft Azure

Search resources

Shift+Space to toggle favorites

app service

App Service Certificates

App Service Domains

App Service Environments

App Service plans

App Services

Keywords: web apps

More services >

Help improve the service menu!

Select the domain to manage.

Access DNS zone

In the domain's left menu, select **DNS zone**.

A screenshot of the Microsoft Azure portal. At the top, it says "kontoso.net" and "App Service Domain". On the left, there's a sidebar with a search bar and links for "Automation script", "Hostname bindings", "Domain renewal", and "DNS zone" (which is highlighted with a red box). The main content area shows "Essentials" with "Resource group: myResourceGroup", "Status: Active", "Location: global", "Subscription name: [redacted]", and "Subscription ID: [redacted]".

This action opens the [DNS zone](#) page of your App Service Domain in Azure DNS. For information on how to edit DNS records, see [How to manage DNS Zones in the Azure portal](#).

Cancel purchase (delete domain)

After you purchase the App Service Domain, you have five days to cancel your purchase for a full refund. After five days, you can delete the App Service Domain, but cannot receive a refund.

Open App Service Domain

In the Azure portal, from the left menu, select **More Services > App Service Domains**.

A screenshot of the Microsoft Azure portal. The left sidebar has a "More services >" button highlighted with a red box. The main area shows a search bar with "app service" typed in. Below the search bar is a list of services: "App Service Certificates", "App Service Domains" (which is highlighted with a red box), "App Service Environments", "App Service plans", and "App Services". There is also a link "Help improve the service menu!" at the bottom right of the search results.

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.

Hostname bindings

Hostnames can be assigned to different azure resources. Here you can find a list of subdomains linked to App Service or Traffic manager.

ROOT OR SUBDOMAIN	ASSIGNED TO	RESOURCE TYPE
www	webapp-custom-dns	Website
@	webapp-custom-dns	Website
abc	webapp-custom-dns	Website

Save Discard

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**.

Hostname bindings

Hostnames can be assigned to different azure resources. Here you can find a list of subdomains linked to App Service or Traffic manager.

ROOT OR SUBDOMAIN	ASSIGNED TO	RESOURCE TYPE
www	webapp-custom-dns	Website
@	webapp-cust... delete ...	Website

Save Discard

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**.

kontoso.net
App Service Domain

Search (Ctrl+ /)

Advanced management Cancel purchase

Overview

Access control (IAM)

Tags

Resource group
myResourceGroup

Status
Active

Domain
http://kontoso.net

Expiration date
July 19, 2018 2:27:28 P

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](#).

More resources

[FAQ: App Service Domain \(preview\) and Custom Domains](#)

Configuring a custom domain name for a web app in Azure App Service using Traffic Manager

11/10/2017 • 7 min 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.

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 4/4/2017 12:46 PM

Type	Name	Value	TTL
NS	@	ns07.domaincontrol.com	1 Hour
NS	@	ns08.domaincontrol.com	1 Hour

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 pre-emptively bind your apps to it, you can create an additional CNAME record. For example, to pre-emptively 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

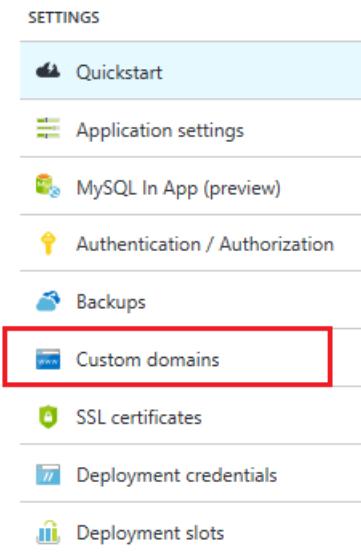
It can take some time for your CNAME to propagate through the DNS system. You can use a service such as <http://www.digwebinterface.com/> to verify that the CNAME is available.

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 Traffic Manager 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 a '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
- Validate:** (button)

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](#).

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Migrate an active DNS name to Azure App Service

1/2/2018 • 3 min 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 Web Apps](#).

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

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:

The screenshot shows a table titled "Records" with two rows of DNS records. The columns are Type, Name, Value, and TTL. Both records are of type NS and point to ns07.domaincontrol.com and ns08.domaincontrol.com respectively, with a TTL of 1 Hour. An "ADD" button is at the bottom right.

Type	Name	Value	TTL
NS	@	ns07.domaincontrol.com	1 Hour
NS	@	ns08.domaincontrol.com	1 Hour

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*.

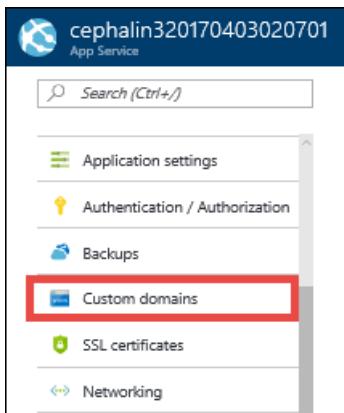
The TXT record you need depends on the DNS record you want to migrate. For examples, see the following table (@ typically represents the root domain):

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.

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**.

Purchased Domains

[Buy Domain](#)

DOMAINS	EXPIRES	STATUS	...
cephalinssl.biz	2018-03-16	Active	...
cephalincontoso.com	2018-04-04	Active	...

External IP Address

Use this IP Address to configure your DNS settings for A records

IP Address:

Hostnames

[Add hostname](#)

HOSTNAMES ASSIGNED TO SITE

cephalin320170403020701.azurewebsites.net

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**.

* Hostname
 

[Validate](#)

Hostname record type

 [CNAME configuration](#)

A CNAME record is used to specify that a domain name is an alias for another domain. [Learn More](#)

CNAME

[Add hostname](#)

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.

HOSTNAMES ASSIGNED TO SITE

www.contoso.com	...
cephalin320170403020701.azurewebsites.net	

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 domains' page in the Azure portal. It lists a purchased domain 'cephalinssl.biz' with an expiration date of '2018-03-16' and a status of 'Active'. Below the domain list, there is a section for 'External IP Address' with a note: 'Use this IP Address to configure your DNS settings for A records'. An input field labeled 'IP Address:' contains '13.84.46.29', which is highlighted with a red box. Further down, there is a 'Hostnames' section with a 'Add hostname' button and a list of assigned hostnames: 'cephalin320170403020701.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
*.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.

Next steps

Learn how to bind a custom SSL certificate to App Service.

[Bind an existing custom SSL certificate to Azure Web Apps](#)

Buy and Configure an SSL Certificate for your Azure App Service

12/7/2017 • 8 min to read • [Edit Online](#)

This tutorial shows you how to secure your web app by purchasing an SSL certificate for your [Azure App Service](#), securely storing it in [Azure Key Vault](#), and associating it with a custom domain.

Step 1 - Log in to Azure

Log in to the Azure portal at <http://portal.azure.com>

Step 2 - Place an SSL Certificate order

You can place an SSL Certificate order by creating a new [App Service Certificate](#) In the [Azure portal](#).

SKU	Price	Features
S1 Standard	\$69.99 USD/YEAR (ESTIMATED)	1 Year, X.509 v3, RSA-SHA256, Auto Renew, Improves SEO
W1 Wild Card	\$299.99 USD/YEAR (ESTIMATED)	1 Year, X.509 v3, RSA-SHA256, Auto Renew, Improves SEO, Wild Card

Note: Create certificate operation may take 1-10 minutes to complete. Once created, App Service Certificates can only be used by other App Services within the same subscription.

Enter a friendly **Name** for your SSL certificate and enter the **Domain Name**

NOTE

This step is one of the most critical parts of the purchase process. Make sure to enter correct host name (custom domain) that you want to protect with this certificate. **DO NOT** append the Host name with WWW.

Select your **Subscription**, **Resource Group**, and **Certificate SKU**

WARNING

App Service Certificates can only be used by other App Services within the same subscription.

Step 3 - Store the certificate in Azure Key Vault

NOTE

Key Vault is an Azure service that helps safeguard cryptographic keys and secrets used by cloud applications and services.

Once the SSL Certificate purchase is complete, you need to open the [App Service Certificates](#) page.

The screenshot displays three windows from the Azure portal:

- ContosoCert (App Service Certificate):** Shows a status message "Configured required Key Vault store" and a "Status" section with "Pending Issuance".
- Certificate Properties (ContosoCert):** Shows certificate details:
 - Distinguished Name: CN=contosocertificate.com
 - Key Size: 2048
 - Product Type: Standard
 - Validity Period: 1 Year(s)
- Certificate Status (ContosoCert):** Shows a step-by-step process:
 - Step 1: Store:** Task: Import certificate into Keyvault for secure administration.
 - Step 2: Verify:** Task: Verify certificate domain ownership.
 - Step 3: Assign:** Task: Certificate ready to use in App Service.

The certificate status is "**Pending Issuance**" as there are few more steps you need to complete before you can start using this certificate.

Click **Certificate Configuration** inside the Certificate Properties page and Click on **Step 1: Store** to store this certificate in Azure Key Vault.

From the **Key Vault Status** page, click **Key Vault Repository** to choose an existing Key Vault to store this certificate **OR Create New Key Vault** to create new Key Vault inside same subscription and resource group.

NOTE

Azure Key Vault has minimal charges for storing this certificate. For more information, see [Azure Key Vault Pricing Details](#).

Once you have selected the Key Vault Repository to store this certificate in, the **Store** option should show success.

Certificate Properties

ContosoCert

Filter settings

SUPPORT + TROUBLESHOOTING

Activity log

CERTIFICATE

Properties

Certificate Configuration

Auto Renew Settings

Rekey and Sync

RESOURCE MANAGEMENT

Tags

Locks

Users

Automation script

Certificate Status

ContosoCert

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.

Step 1: Store

Certificate successfully imported to Key vault.

Step 2: Verify

Verify certificate domain ownership

Step 3: Assign

Certificate ready to use in App Service

Step 4 - Verify the Domain Ownership

From the same **Certificate Configuration** page you used in Step 3, click **Step 2: Verify**.

Choose the preferred domain verification method.

There are four types of domain verification supported by App Service Certificates: App Service, Domain, Mail, and Manual Verification. These verification types are explained in more details in the [Advanced section](#).

NOTE

App Service Verification is the most convenient option when the domain you want to verify 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.

Click on **Verify** button to complete this step.

Certificate Status

ContosoCert

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.

Step 1: Store

Certificate successfully imported to Key vault.

Step 2: Verify

Verify certificate domain ownership

Step 3: Assign

Certificate ready to use in App Service

Domain Verification

ContosoCert

Verify Refresh

Domain Verification Pending. Verify operations may take 5-10 minutes to take effect.

Domain Verification Token:

Select the method to verify domain ownership:

Domain Registration

Domain registration method allows you to create a txt record directly on your domain and allow the certificate authority to verify your ownership of the domain.

DOMAIN REGISTRATION HOST NAME: contosocertificate.com

After clicking **Verify**, use the **Refresh** button until the **Verify** option should show success.

The screenshot shows two adjacent windows from the Azure portal. The left window is titled 'Certificate Properties' for 'ContosoCert'. It has a sidebar with sections like 'SUPPORT + TROUBLESHOOTING' (Activity log), 'CERTIFICATE' (Properties, Certificate Configuration, Auto Renew Settings, Rekey and Sync), and 'RESOURCE MANAGEMENT' (Tags, Locks, Users, Automation script). The right window is titled 'Certificate Status' for 'ContosoCert'. It contains a note: '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 are three steps: 'Step 1: Store' (key icon, checked), 'Step 2: Verify' (globe icon, checked, highlighted with a red border), and 'Step 3: Assign' (cloud icon, checked).

Step 5 - Assign Certificate to App Service App

NOTE

Before performing the steps in this section, you must have associated a custom domain name with your app. For more information, see [Configuring a custom domain name for a web app](#).

In the [Azure portal](#), click the **App Service** option on the left of the page.

Click the name of your app to which you want to assign this certificate.

In the **Settings**, click **SSL certificates**.

Click **Import App Service Certificate** and select the certificate that you just purchased.

Search (Ctrl+/)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

APP SERVICE PLAN

- App Service plan
- Scale up (App Service plan)
- Scale out (App Service plan)
- Quotas
- Change App Service plan

SETTINGS

- Quickstart
- Application settings
- MySQL In App (preview)
- Authentication / Authorization
- Backups
- Custom domains
- SSL certificates**
- Deployment credentials
- Deployment slots

SSL

Certificates must be associated with your application before you can use them to create a binding. You can upload a certificate you purchased externally, or import an App Service Certificate.

Import App Service Certificate

Upload Certificate

You have no certificates. Upload a certificate now to get started.

SSL bindings

Add binding

HOST NAME	CERTIFICATE	SSL TYPE
No results		

In the **ssl bindings** section Click on **Add bindings**, and use the dropdowns to select the domain name to secure with SSL, and the certificate to use. You may also select whether to use **Server Name Indication (SNI)** or IP based SSL.

Add SSL Binding



SSL bindings

Use the dropdowns to select the Hostname to secure with SSL, and the certificate to use. You may also select whether to use Server Name Indication (SNI) or IP based SSL.
[Learn more](#)

* Hostname

www.contosocertificate.com



* Certificate

contosocertificate.com,www.contosocertificate.com (



* SSL Type:

SNI SSL



Add Binding

Click **Add Binding** to save the changes and enable SSL.

NOTE

If you selected **IP based SSL** and your custom domain is configured using an A record, you must perform the following additional steps. These are explained in more details in the [Advanced section](#).

At this point, you should be able to visit your app using `HTTPS://` instead of `HTTP://` to verify that the certificate has been configured correctly.

Step 6 - Management tasks

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-AzureRmResourceGroup -Name $webappname -Location $location

# Create an App Service plan in Free tier.
New-AzureRmAppServicePlan -Name $webappname -Location $location ` 
-ResourceGroupName $webappname -Tier Free

# Create a web app.
New-AzureRmWebApp -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-AzureRmAppServicePlan -Name $webappname -ResourceGroupName $webappname ` 
-Tier Basic

# Add a custom domain name to the web app.
Set-AzureRmWebApp -Name $webappname -ResourceGroupName $webappname ` 
-HostNames @($fqdn,"$webappname.azurewebsites.net")

# Upload and bind the SSL certificate to the web app.
New-AzureRmWebAppSSLBinding -WebAppName $webappname -ResourceGroupName $webappname -Name $fqdn ` 
-CertificateFilePath $pfxPath -CertificatePassword $pfxPassword -SslState SniEnabled

```

Advanced

Verifying Domain Ownership

There are two more types of domain verification supported by App service Certificates: Mail, and Manual Verification.

Mail Verification

Verification email has already been sent to the Email Address(es) associated with this custom domain. To complete the Email verification step, open the email and click the verification link.

Domain Access Approval

A Certificate Signing Request (CSR) has been submitted for the following domain(s).
Please indicate your approval of the certificate request. If you do not approve the request, a certificate cannot be issued for the listed domain(s).

Approve **Disapprove**

[Legal Docs](#) • Copyright © 1999-2017 GoDaddy.com, LLC All rights reserved.

If you need to resend the verification email, click the **Resend Email** button.

Domain Verification

Choose this option only for an [App Service domain that you purchased from Azure](#). Azure automatically adds the verification TXT record for you and completes the process.

Manual Verification

IMPORTANT

HTML Web Page Verification (only works with Standard Certificate SKU)

1. Create an HTML file named "**starfield.html**"
2. Content of this file should be the exact name of the Domain Verification Token. (You can copy the token from the Domain Verification Status page)
3. Upload this file at the root of the web server hosting your domain
`/well-known/pki-validation/starfield.html`
4. Click **Refresh** to update the certificate status after verification is completed. It might take few minutes for verification to complete.

TIP

Verify in a terminal using `curl -G http://<domain>/well-known/pki-validation/starfield.html` the response should contain the `<verification-token>`.

DNS TXT Record Verification

1. Using your DNS manager, Create a TXT record on the `@` subdomain with value equal to the Domain Verification Token.
2. Click "**Refresh**" to update the Certificate status after verification is completed.

TIP

You need to create a TXT record on `@.<domain>` with value `<verification-token>`.

Assign Certificate to App Service App

If you selected **IP based SSL** and your custom domain is configured using an A record, you must perform the following additional steps:

After you have configured an IP based SSL binding, a dedicated IP address is assigned to your app. You can find this IP address on the **Custom domain** page under settings of your app, right above the **Hostnames** section. It is listed as **External IP Address**

External Ip Address

Use this IP Address to configure your DNS settings for A Record hostnames

IP Address: **104.40.28.133**

Hostnames

Add hostname

This IP address is different than the virtual IP address used previously to configure the A record for your domain. If you are configured to use SNI based SSL, or are not configured to use SSL, no address is listed for this entry.

Using the tools provided by your domain name registrar, modify the A record for your custom domain name to point to the IP address from the previous step.

Rekey and Sync the Certificate

If you ever need to rekey your certificate, select the **Rekey and Sync** option from the **Certificate Properties** page.

Click **Rekey** Button to initiate the process. This process can take 1-10 minutes to complete.

The screenshot shows two side-by-side windows from the Azure portal. The left window is titled 'Certificate Properties' for 'ContosoCert'. It has a sidebar with 'Activity log', 'Properties', 'Certificate Configuration', 'Auto Renew Settings', and 'Rekey and Sync' (which is highlighted with a red box). Below the sidebar are sections for 'Support + Troubleshooting' (Activity log), 'CERTIFICATE' (Properties, Certificate Configuration, Auto Renew Settings), and 'RESOURCE MANAGEMENT' (Tags, Locks, Users, Automation script). The right window is titled 'Rekey and Sync' for 'ContosoCert'. It has a 'Rekey Certificate' section with a 'Rekey' button. Below it is a 'Current Certificate Thumbprint' field (which is also highlighted with a red box) and a table with columns 'LINKED CERTIFICA...', 'RESOUCE GROUP', and 'THUMBPRINT'. The table shows 'No associated resource found.'

Rekeying your certificate rolls the certificate with a new certificate issued from the certificate authority.

Why is my SSL certificate not auto-renewed?

If your SSL certificate is configured for auto-renewal, but it is not automatically renewed, you may have a pending domain verification. Note the following:

- GoDaddy, which generates App Service certificates, requires domain verification once every three years. The domain administrator receives an email once every three years to verify the domain. Failure to check the email or verify your domain prevents the App Service certificate from being automatically renewed.
- All App Service certificates issued prior to March 31 2017 require reverification of domain at the time of next renewal (even if the auto-renewal is enabled for the certificate). This is a result of change in GoDaddy policy. Check your email and complete this one-time domain verification to continue the auto-renewal of the App Service certificate.

More resources

- [Use an SSL certificate in your application code in Azure App Service](#)
- [FAQ : App Service Certificates](#)

Configure your App Service app to use Azure Active Directory login

11/22/2017 • 5 min to read • [Edit Online](#)

This article shows you how to configure Azure App Services to use Azure Active Directory as an authentication provider.

Configure Azure Active Directory using express settings

1. In the [Azure portal](#), navigate to your App Service app. In the left navigation, select **Authentication / Authorization**.
2. If **Authentication / Authorization** is not enabled, select **On**.
3. Select **Azure Active Directory**, and then select **Express** under **Management Mode**.
4. Select **OK** to register the App Service app in Azure Active Directory. This creates a new app registration. If you want to choose an existing app registration instead, click **Select an existing app** and then search for the name of a previously created app registration within your tenant. Click the app registration to select it and click **OK**. Then click **OK** on the Azure Active Directory settings page. By default, App Service provides authentication but does not restrict authorized access to your site content and APIs. You must authorize users in your app code.
5. (Optional) To restrict access to your site to only users authenticated by Azure Active Directory, set **Action to take when request is not authenticated** to **Log in with Azure Active Directory**. This requires that all requests be authenticated, and all unauthenticated requests are redirected to Azure Active Directory for authentication.
6. Click **Save**.

You are now ready to use Azure Active Directory for authentication in your App Service app.

(Alternative method) Manually configure Azure Active Directory with advanced settings

You can also choose to provide configuration settings manually. This is the preferred solution if the AAD tenant you wish to use is different from the tenant with which you sign into Azure. To complete the configuration, you must first create a registration in Azure Active Directory, and then you must provide some of the registration details to App Service.

Register your App Service app with Azure Active Directory

1. Log on to the [Azure portal](#), and navigate to your App Service app. Copy your app **URL**. You will use this to configure your Azure Active Directory app registration.
2. Navigate to **Active Directory**, then select the **App registrations**, then click **New application registration** at the top to start a new app registration.
3. In the **Create** page, enter a **Name** for your app registration, select the **Web App / API** type, in the **Sign-on URL** box paste the application URL (from step 1). Then click to **Create**.
4. In a few seconds, you should see the new app registration you just created.
5. Once the app registration has been added, click on the app registration name, click on **Settings** at the top, then click on **Properties**
6. In the **App ID URI** box, paste in the Application URL (from step 1), also in the **Home Page URL** paste in the Application URL (from step 1) as well, then click **Save**
7. Now click on the **Reply URLs**, edit the **Reply URL**, paste in the Application URL (from step 1), modify the

protocol to make sure you have **https://** protocol (not http://), then appended to the end of the URL, `/auth/login/aad/callback` (For example, `https://contoso.azurewebsites.net/.auth/login/aad/callback`). Click **Save**.

8. At this point, copy the **Application ID** for the app. Keep it for later use. You will need it to configure your App Service app.
9. Close the **Registered app** page. On the **App registrations** page, click on the **Endpoints** button at the top, then copy the **Federation Metadata Document** URL.
10. Open a new browser window and navigate to the URL by pasting and browsing to the XML page. At the top of document is an **EntityDescriptor** element, there should be an **entityID** attribute of the form `https://sts.windows.net/` followed by a GUID specific to your tenant (called a "tenant ID"). Copy this value - it serves as your **Issuer URL**. You will configure your application to use it later.

Add Azure Active Directory information to your App Service app

1. Back in the [Azure portal](#), navigate to your App Service app. Click **Authentication/Authorization**. If the Authentication/Authorization feature is not enabled, turn the switch to **On**. Click on **Azure Active Directory**, under Authentication Providers, to configure your app. (Optional) By default, App Service provides authentication but does not restrict authorized access to your site content and APIs. You must authorize users in your app code. Set **Action to take when request is not authenticated** to **Log in with Azure Active Directory**. This option requires that all requests be authenticated, and all unauthenticated requests are redirected to Azure Active Directory for authentication. 2. In the Active Directory Authentication configuration, click **Advanced** under **Management Mode**. Paste the Application ID into the Client ID box (from step 8) and paste in the entityId (from step 10) into the Issuer URL value. Then click **OK**.
2. On the Active Directory Authentication configuration page, click **Save**.

You are now ready to use Azure Active Directory for authentication in your App Service app.

(Optional) Configure a native client application

Azure Active Directory also allows you to register native clients, which provides greater control over permissions mapping. You need this if you wish to perform logins using a library such as the **Active Directory Authentication Library**.

1. Navigate to **Azure Active Directory** in the [Azure portal](#).
2. In the left navigation, select **App registrations**. Click **New app registration** at the top.
3. In the **Create** page, enter a **Name** for your app registration. Select **Native** in **Application type**.
4. In the **Redirect URI** box, enter your site's `/auth/login/done` endpoint, using the HTTPS scheme. This value should be similar to <https://contoso.azurewebsites.net/.auth/login/done>. If creating a Windows application, instead use the **package SID** as the URI.
5. Click **Create**.
6. Once the app registration has been added, select it to open it. Find the **Application ID** and make a note of this value.
7. Click **All settings > Required permissions > Add > Select an API**.
8. Type the name of the App Service app that you registered earlier to search for it, then select it and click **Select**.
9. Select **Access <app_name>**. Then click **Select**. Then click **Done**.

You have now configured a native client application that can access your App Service app.

Related Content

- [App Service Authentication / Authorization overview](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#)

Learn how to add App Service authentication to your mobile app.

How to configure your App Service application to use Facebook login

9/19/2017 • 2 min to read • [Edit Online](#)

This topic shows you how to configure Azure App Service to use Facebook as an authentication provider.

To complete the procedure in this topic, you must have 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. Log on to the [Azure portal](#), and navigate to your application. Copy your **URL**. You will use this to configure your Facebook app.
2. In another browser window, navigate to the [Facebook Developers](#) website and sign-in with your Facebook account credentials.
3. (Optional) If you have not already registered, click **Apps > Register as a Developer**, then accept the policy and follow the registration steps.
4. Click **My Apps > Add a New App > Website > Skip and Create App ID**.
5. In **Display Name**, type a unique name for your app, type your **Contact Email**, choose a **Category** for your app, then click **Create App ID** and complete the security check. This takes you to the developer dashboard for your new Facebook app.
6. Under "Facebook Login," click **Get Started**. Add your application's **Redirect URI** to **Valid OAuth redirect URLs**, then click **Save Changes**.

NOTE

Your redirect URI is the URL of your application appended with the path, `./auth/login/facebook/callback`. For example, `https://contoso.azurewebsites.net/.auth/login/facebook/callback`. Make sure that you are using the HTTPS scheme.

7. In the left-hand navigation, click **Settings**. On the **App Secret** field, click **Show**, provide your password if requested, then make a note of the values of **App ID** and **App Secret**. You use these later to configure your application in Azure.

IMPORTANT

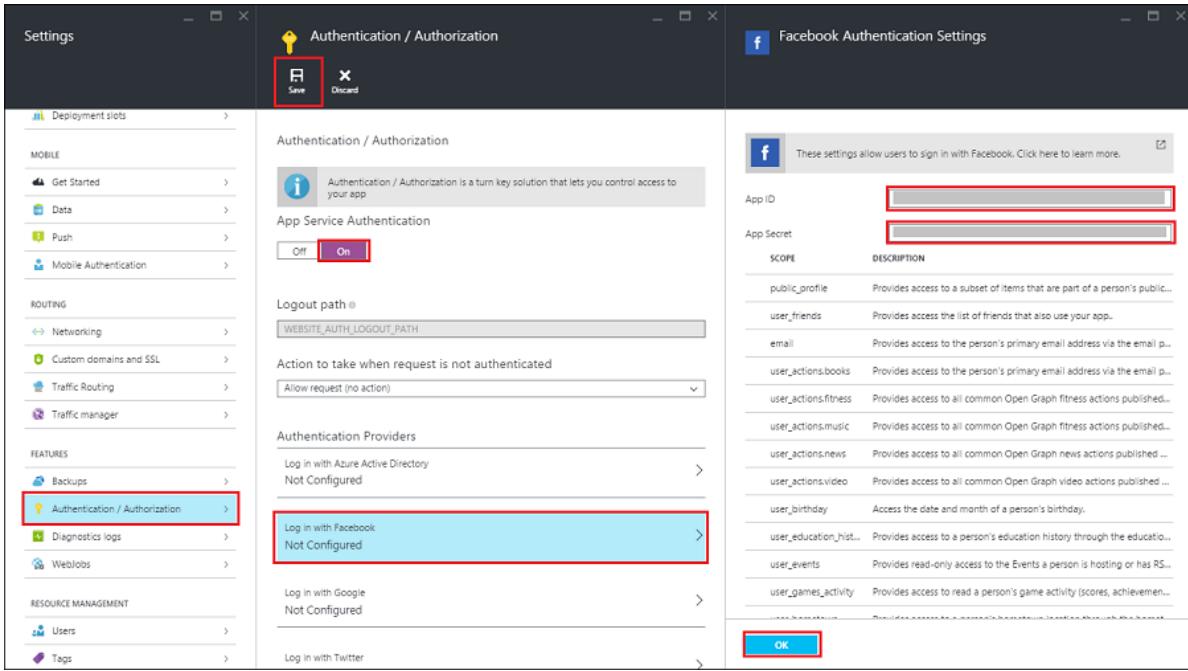
The app secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

8. The Facebook account which was used to register the application is an administrator of the app. At this point, only administrators can sign into this application. To authenticate other Facebook accounts, click **App Review** and enable **Make public** to enable general public access using Facebook authentication.

Add Facebook information to your application

1. Back in the [Azure portal](#), navigate to your application. Click **Settings > Authentication / Authorization**, and make sure that **App Service Authentication** is **On**.
2. Click **Facebook**, paste in the App ID and App Secret values which you obtained previously, optionally

enable any scopes needed by your application, then click **OK**.



By default, App Service provides authentication but does not restrict authorized access to your site content and APIs. You must authorize users in your app code.

3. (Optional) To restrict access to your site to only users authenticated by Facebook, set **Action to take when request is not authenticated** to **Facebook**. This requires that all requests be authenticated, and all unauthenticated requests are redirected to Facebook for authentication.
4. When done configuring authentication, click **Save**.

You are now ready to use Facebook for authentication in your app.

Related Content

- [App Service Authentication / Authorization overview](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#)

Learn how to add App Service authentication to your mobile app.

How to configure your App Service application to use Google login

9/19/2017 • 2 min 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

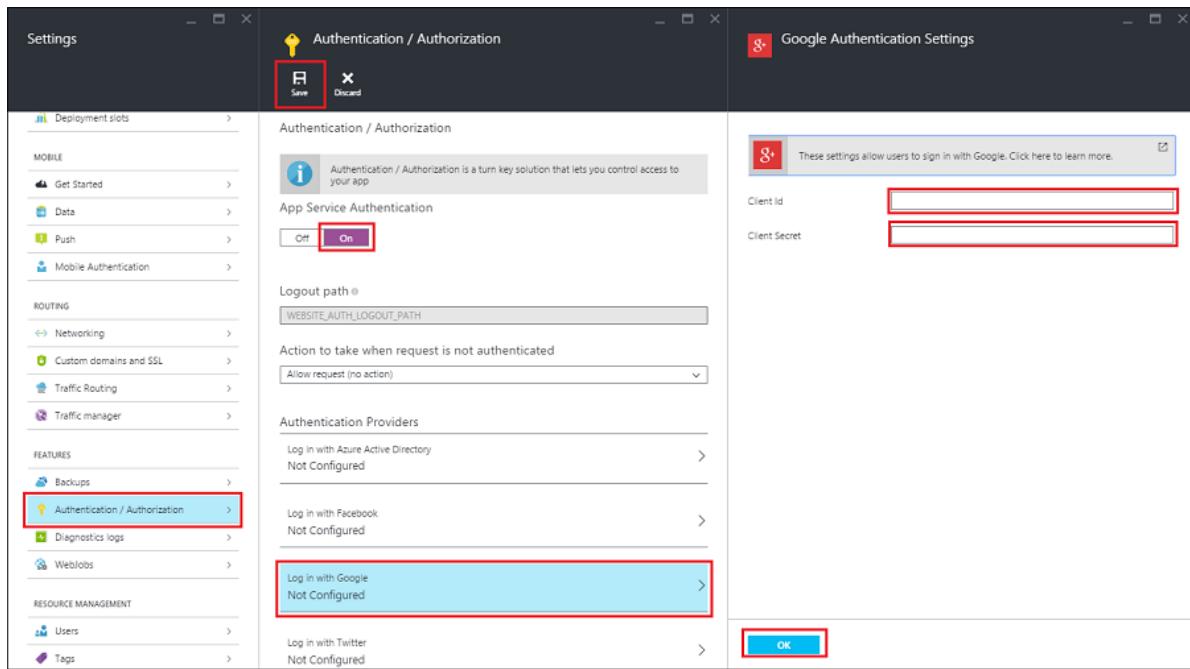
1. Log on to the [Azure portal](#), and navigate to your application. Copy your **URL**, which you use later to configure your Google app.
2. Navigate to the [Google apis](#) website, sign in with your Google account credentials, click **Create Project**, provide a **Project name**, then click **Create**.
3. Under **Social APIs** click **Google+ API** and then **Enable**.
4. In the left navigation, **Credentials** > **OAuth consent screen**, then select your **Email address**, enter a **Product Name**, and click **Save**.
5. In the **Credentials** tab, click **Create credentials** > **OAuth client ID**, then select **Web application**.
6. Paste the App Service **URL** you copied earlier into **Authorized JavaScript Origins**, then paste your redirect URI into **Authorized Redirect URI**. The redirect URI is the URL of your application appended with the path, `/auth/login/google/callback`. For example, `https://contoso.azurewebsites.net/.auth/login/google/callback`. Make sure that you are using the HTTPS scheme. Then click **Create**.
7. On the next screen, make a note of the values of the client ID and client secret.

IMPORTANT

The client 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. Back in the [Azure portal](#), navigate to your application. Click **Settings**, and then **Authentication / Authorization**.
2. If the Authentication / Authorization feature is not enabled, turn the switch to **On**.
3. Click **Google**. Paste in the App ID and App Secret values which you obtained previously, and optionally enable any scopes your application requires. Then click **OK**.



By default, App Service provides authentication but does not restrict authorized access to your site content and APIs. You must authorize users in your app code.

4. (Optional) To restrict access to your site to only users authenticated by Google, set **Action to take when request is not authenticated** to **Google**. This requires that all requests be authenticated, and all unauthenticated requests are redirected to Google for authentication.
5. Click **Save**.

You are now ready to use Google for authentication in your app.

Related Content

- [App Service Authentication / Authorization overview](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#)

Learn how to add App Service authentication to your mobile app.

How to configure your App Service application to use Microsoft Account login

9/19/2017 • 1 min to read • [Edit Online](#)

This topic shows you how to configure Azure App Service to use Microsoft Account as an authentication provider.

Register your app with Microsoft Account

1. Log on to the [Azure portal](#), and navigate to your application. Copy your **URL**, which later you use to configure your app with Microsoft Account.
2. Navigate to the [My Applications](#) page in the Microsoft Account Developer Center, and log on with your Microsoft account, if required.
3. Click **Add an app**, then type an application name, and click **Create application**.
4. Make a note of the **Application ID**, as you will need it later.
5. Under "Platforms," click **Add Platform** and select "Web".
6. Under "Redirect URIs" supply the endpoint for your application, then click **Save**.

NOTE

Your redirect URI is the URL of your application appended with the path, `/.auth/login/microsoftaccount/callback`.

For example, `https://contoso.azurewebsites.net/.auth/login/microsoftaccount/callback`.

Make sure that you are using the HTTPS scheme.

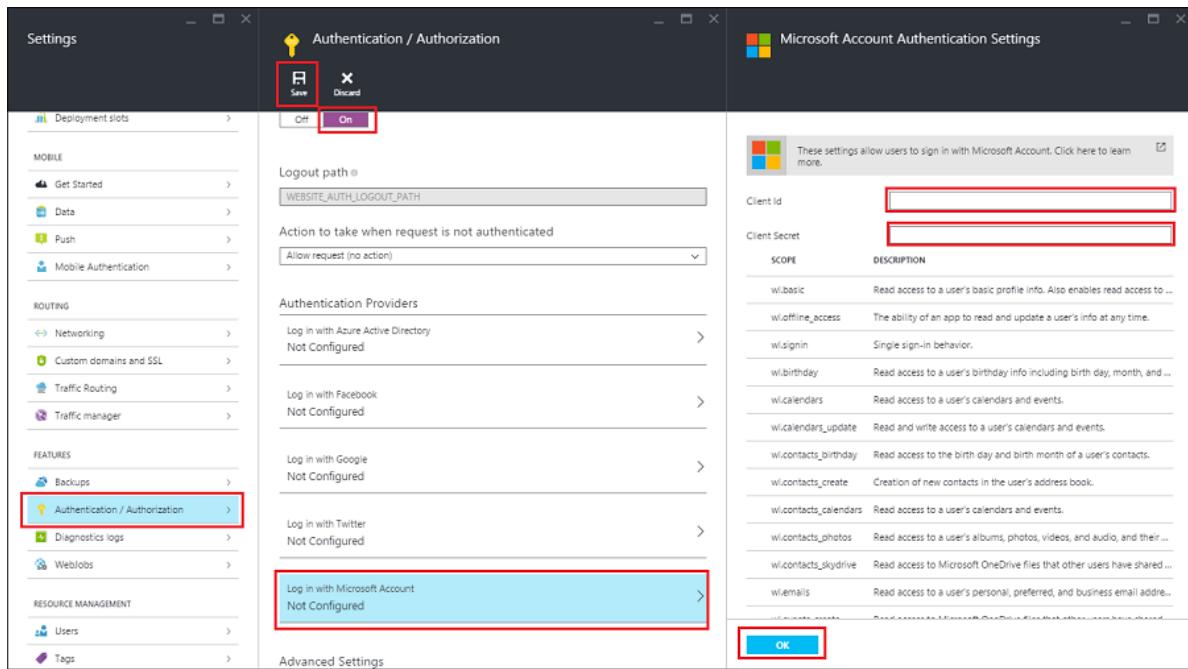
7. Under "Application Secrets," click **Generate New Password**. Make note of the value that appears. Once you leave the page, it will not be displayed again.

IMPORTANT

The 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. Back in the [Azure portal](#), navigate to your application, click **Settings > Authentication / Authorization**.
2. If the Authentication / Authorization feature is not enabled, switch it **On**.
3. Click **Microsoft Account**. Paste in the Application ID and Password values which you obtained previously, and optionally enable any scopes your application requires. Then click **OK**.



By default, App Service provides authentication but does not restrict authorized access to your site content and APIs. You must authorize users in your app code.

4. (Optional) To restrict access to your site to only users authenticated by Microsoft account, set **Action to take when request is not authenticated** to **Microsoft Account**. This requires that all requests be authenticated, and all unauthenticated requests are redirected to Microsoft account for authentication.
5. Click **Save**.

You are now ready to use Microsoft Account for authentication in your app.

Related content

- [App Service Authentication / Authorization overview](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#)

Learn how to add App Service authentication to your mobile app.

How to configure your App Service application to use Twitter login

9/19/2017 • 2 min to read • [Edit Online](#)

This topic shows you how to configure Azure App Service to use Twitter as an authentication provider.

To complete the procedure in this topic, you must have 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

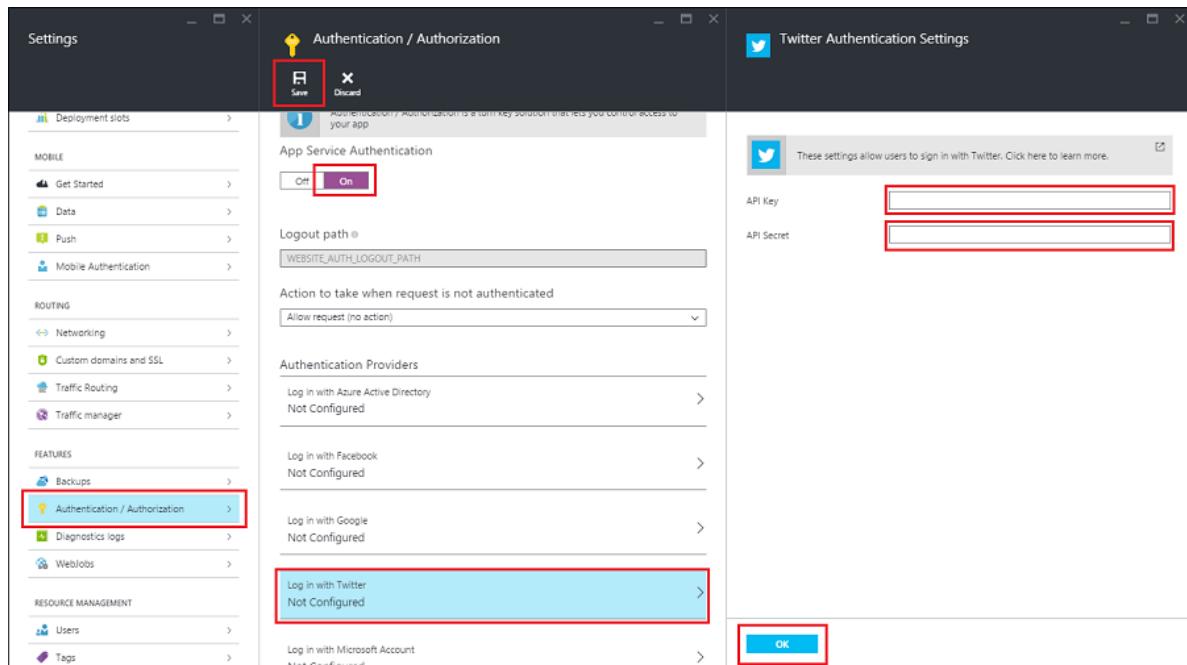
1. Log on to the [Azure portal](#), and navigate to your application. Copy your **URL**. You will use this to configure your Twitter app.
2. Navigate to the [Twitter Developers](#) website, sign in with your Twitter account credentials, and click **Create New App**.
3. Type in the **Name** and a **Description** for your new app. Paste in your application's **URL** for the **Website** value. Then, for the **Callback URL**, paste the **Callback URL** you copied earlier. This is your Mobile App gateway appended with the path, `/auth/login/twitter/callback`. For example,
`https://contoso.azurewebsites.net/.auth/login/twitter/callback`. Make sure that you are using the HTTPS scheme.
4. At the bottom of the page, read and accept the terms. Then click **Create your Twitter application**. This registers the app and displays the application details.
5. Click the **Settings** tab, check **Allow this application to be used to sign in with Twitter**, then click **Update Settings**.
6. Select the **Keys and Access Tokens** tab. Make a note of the values of **Consumer Key (API Key)** and **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. Back in the [Azure portal](#), navigate to your application. Click **Settings**, and then **Authentication / Authorization**.
2. If the Authentication / Authorization feature is not enabled, turn the switch to **On**.
3. Click **Twitter**. Paste in the App ID and App Secret values which you obtained previously. Then click **OK**.



By default, App Service provides authentication but does not restrict authorized access to your site content and APIs. You must authorize users in your app code.

4. (Optional) To restrict access to your site to only users authenticated by Twitter, set **Action to take when request is not authenticated** to **Twitter**. This requires that all requests be authenticated, and all unauthenticated requests are redirected to Twitter for authentication.
5. Click **Save**.

You are now ready to use Twitter for authentication in your app.

Related Content

- [App Service Authentication / Authorization overview](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#)

Learn how to add App Service authentication to your mobile app.

How to use Azure Managed Service Identity (public preview) in App Service and Azure Functions

12/6/2017 • 6 min to read • [Edit Online](#)

NOTE

Managed Service Identity for App Service and Azure Functions is currently in preview.

This topic shows you how to create a managed app identity for App Service and Azure Functions applications and how to use it to access other resources. A managed service identity from Azure Active Directory 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 Service Identity, see the [Managed Service Identity overview](#).

Creating an app with an identity

Creating an app with an identity requires an additional property to be set on the application.

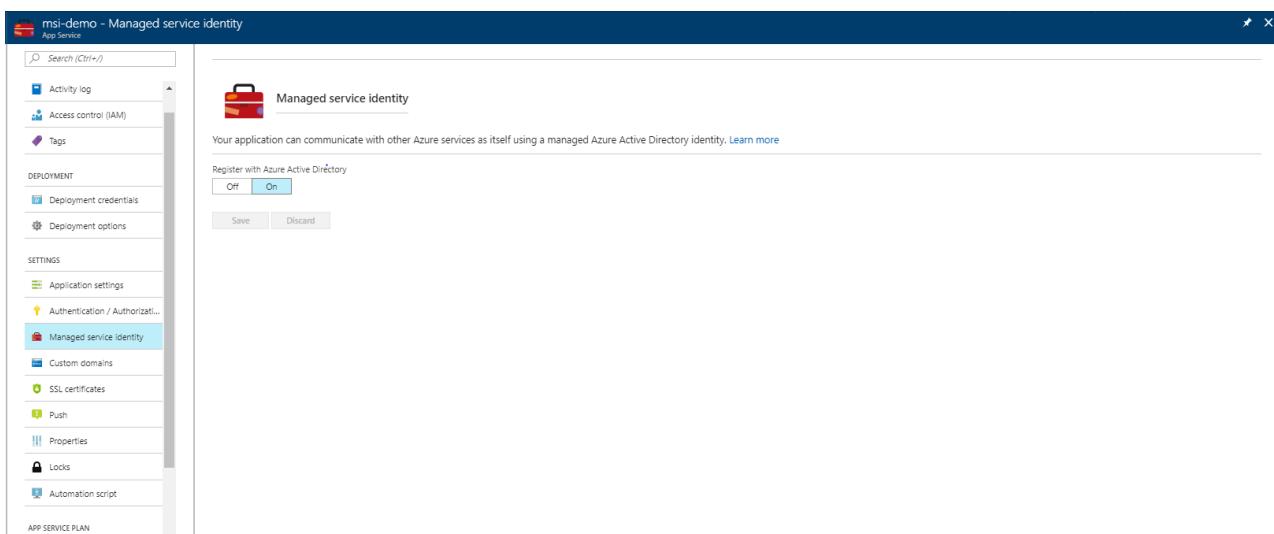
NOTE

Only the primary slot for a site will receive the identity. Managed service identities for deployment slots are not yet supported.

Using the Azure portal

To set up a managed service 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 **Managed service identity**.
4. Switch **Register with Azure Active Directory** to **On**. Click **Save**.



Using the Azure CLI

To set up a managed service identity using the Azure CLI, you will need to use the `az webapp assign-identity` 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 CLI 2.0](#) (2.0.21 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 `assign-identity` command to create the identity for this application:

```
az webapp assign-identity --name myApp --resource-group 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"  
}
```

This 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": {
    "tenantId": "<TENANTID>",
    "principalId": "<PRINCIPALID>"
}
```

Where `<TENANTID>` and `<PRINCIPALID>` are replaced with GUIDs. The `tenantId` property identifies what AAD tenant the application belongs to. The `principalId` is a unique identifier for the application's new identity. Within AAD, the application has the same name that you gave to your App Service or Azure Functions instance.

Obtaining tokens for Azure resources

An app can use its identity to get tokens to 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 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 Managed Service Identity 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. For .NET applications, the `Microsoft.Azure.Services.AppAuthentication` library provides an abstraction over this protocol and supports a local development experience.

Using the `Microsoft.Azure.Services.AppAuthentication` library for .NET

For .NET applications and functions, the simplest way to work with a managed service 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 2.0](#), 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 [Microsoft.Azure.KeyVault](#) NuGet packages to your application.

2. Add the following code to your application:

```
using Microsoft.Azure.Services.AppAuthentication;
using Microsoft.Azure.KeyVault;
// ...
var azureServiceTokenProvider = new AzureServiceTokenProvider();
string accessToken = await azureServiceTokenProvider.GetAccessTokenAsync("https://management.azure.com/");
// 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 REST protocol

An app with a managed service identity has two environment variables defined:

- MSI_ENDPOINT
- MSI_SECRET

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.
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.

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 Managed Service 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

To make this request in C#:

```
public static async Task<HttpResponseMessage> GetToken(string resource, string apiversion) {
    HttpClient client = new HttpClient();
    client.DefaultRequestHeaders.Add("Secret", Environment.GetEnvironmentVariable("MSI_SECRET"));
    return await client.GetAsync(String.Format("{0}/?resource={1}&api-version={2}",
    Environment.GetEnvironmentVariable("MSI_ENDPOINT"), resource, apiversion));
}
```

TIP

For .NET languages, you can also use [Microsoft.Azure.Services.AppAuthentication](#) instead of crafting this request yourself.

In NodeJS:

```
const rp = require('request-promise');
const getToken = function(resource, apiver, cb) {
    var options = {
        uri: `${process.env["MSI_ENDPOINT"]}/?resource=${resource}&api-version=${apiver}`,
        headers: {
            'Secret': process.env["MSI_SECRET"]
        }
    };
    rp(options)
        .then(cb);
}
```

In PowerShell:

```
$apiVersion = "2017-09-01"
$resourceURI = "https://<AAD-resource-URI-for-resource-to-obtain-token>"
$tokenAuthURI = $env:MSI_ENDPOINT + "?resource=$resourceURI&api-version=$apiVersion"
$tokenResponse = Invoke-RestMethod -Method Get -Headers @{"Secret"="$env:MSI_SECRET"} -Uri $tokenAuthURI
$accessToken = $tokenResponse.access_token
```

Use an SSL certificate in your application code in Azure App Service

12/1/2017 • 2 min to read • [Edit Online](#)

This how-to guide shows how to use one of SSL certificates you have uploaded or imported into your App Service app in your application code. An example of the use case is that your app accesses an external service that requires certificate authentication.

This approach to using SSL certificates in your code makes use of the SSL functionality in App Service, which requires your app to be in **Basic** tier or above. An alternative is to include the certificate file in your application directory and load it directly (see [Alternative: load certificate as a file](#)). However, this alternative does not let you hide the private key in the certificate from the application code or the developer. Furthermore, if your application code is in an open source repository, keeping a certificate with a private key in the repository is not an option.

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 complete this how-to guide:

- [Create an App Service app](#)
- [Map a custom DNS name to your web app](#)
- [Upload an SSL certificate or import an App Service Certificate](#) to your web app

Load your certificates

To use a certificate that is uploaded to or imported into App Service, first make it accessible to your application code. You do this with the `WEBSITE_LOAD_CERTIFICATES` app setting.

In the [Azure portal](#), open your web app page.

In the left navigation, click **SSL certificates**.

NAME	EXPIRATION	THUMBPRINT
contoso.com,www.c...	4/20/2018	[REDACTED]

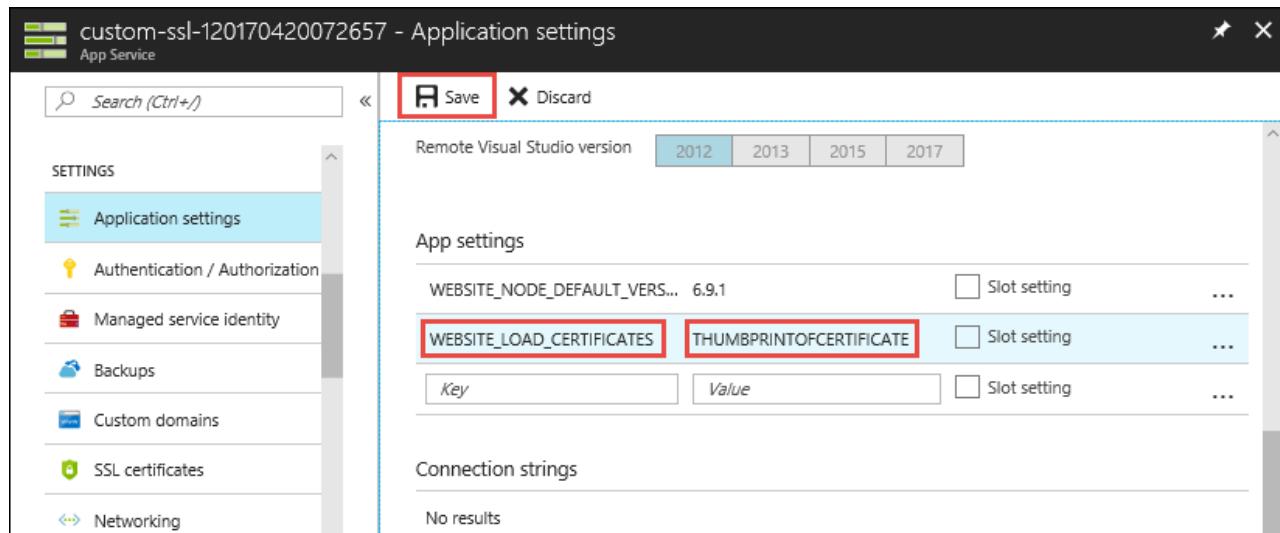
HOST NAME	CERTIFICATE	SSL TYPE
No results		

All your uploaded and imported SSL certificates for this web app are shown here with their thumbprints. Copy the

thumbprint of the certificate you want to use.

In the left navigation, click **Application settings**.

Add an app setting called `WEBSITE_LOAD_CERTIFICATES` and set its value to the thumbprint of the certificate. To make multiple certificates accessible, use comma-separated thumbprint values. To make all certificates accessible, set the value to `*`.



The screenshot shows the 'Application settings' page for an Azure App Service named 'custom-ssl-120170420072657'. The left sidebar lists various settings like Application settings, Authentication / Authorization, Managed service identity, Backups, Custom domains, SSL certificates, and Networking. The main area shows 'App settings' with two entries: 'WEBSITE_NODE_DEFAULT_VERS...' (value: 6.9.1) and 'WEBSITE_LOAD_CERTIFICATES' (value: THUMBPRINTOFCERTIFICATE). Both of these entries have a red box drawn around them. Below these are 'Connection strings' and a note 'No results'.

When finished, click **Save**.

The configured certificate is now ready to be used by your code.

Use certificate in C# code

Once your certificate is accessible, you access it in C# code 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();
...
```

Alternative: load certificate as a file

This section shows how to and load a certificate file that is in your application directory.

The following C# example loads a certificate called `mycert.pfx` from the `certs` directory of your app's repository.

```
using System;
using System.Security.Cryptography.X509Certificates;

...
// Replace the parameter with "~/<relative-path-to-cert-file>".
string certPath = Server.MapPath("~/certs/mycert.pfx");

X509Certificate2 cert = GetCertificate(certPath, signatureBlob.Thumbprint);
...
```

How To Configure TLS Mutual Authentication for Web App

9/19/2017 • 4 min to read • [Edit Online](#)

Overview

You can restrict access to your Azure web app by enabling different types of authentication for it. One way to do so is to authenticate using a client certificate when the request is over TLS/SSL. This mechanism is called TLS mutual authentication or client certificate authentication and this article will detail how to setup your web 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.

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

Configure Web App for Client Certificate Authentication

To setup your web app to require client certificates you need to add the clientCertEnabled site setting for your web app and set it to true. This setting is not currently available through the management experience in the Portal, and the REST API will need to be used to accomplish this.

You can use the [ARMClient tool](#) to make it easy to craft the REST API call. After you log in with the tool you will need to issue the following command:

```
ARMClient PUT subscriptions/{Subscription Id}/resourcegroups/{Resource Group Name}/providers/Microsoft.Web/sites/{Website Name}?api-version=2015-04-01 @enableclientcert.json -verbose
```

replacing everything in {} with information for your web app and creating a file called enableclientcert.json with the following JSON content:

```
{
  "location": "My Web App Location",
  "properties": {
    "clientCertEnabled": true
  }
}
```

Make sure to change the value of "location" to wherever your web app is located e.g. North Central US or West US etc.

You can also use <https://resources.azure.com> to flip the `clientCertEnabled` property to `true`.

Note: If you run ARMClient from Powershell, you will need to escape the @ symbol for the JSON file with a back tick `.

Accessing the Client Certificate From Your Web App

If you are using ASP.NET and configure your app to use client certificate authentication, the certificate will be available through the **HttpRequest.ClientCertificate** property. For other application stacks, the client cert will be available in your app through a base64 encoded value in the "X-ARR-ClientCert" request header. Your application can create a certificate from this value and then use it for authentication and authorization purposes in your application.

Special Considerations for Certificate Validation

The client certificate that is sent to the application does not go through any validation by the Azure Web Apps platform. Validating this certificate is the responsibility of the web app. Here is sample ASP.NET code that validates certificate properties for authentication purposes.

```
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.ToString(" ") +
certificate.NotBefore.ToString(" ") +
                    certExpiryDate = certificate.NotAfter.ToString(" ") +
certificate.NotAfter.ToString(" ");
                }
                catch (Exception ex)
                {
                    errorString = ex.ToString();
                }
                finally
                {
                    isValidCert = IsValidClientCertificate();
                }
            }
        }
    }
}
```

```

        if (!isValidCert) response.StatusCode = 403;
        else Response.StatusCode = 200;
    }
}
else
{
    certHeader = "";
}
}

// This is a SAMPLE verification routine. Depending on your application logic and security
requirements,
// you should modify this method
//
private bool IsValidClientCertificate()
{
    // In this example we will only accept the certificate as a valid certificate if all the
conditions below are met:
    // 1. The certificate is not expired and is active for the current time on server.
    // 2. The subject name of the certificate has the common name nildevecc
    // 3. The issuer name of the certificate has the common name nildevecc and organization name
Microsoft Corp
    // 4. The thumbprint of the certificate is 30757A2E831977D8BD9C8496E4C99AB26CB9622B
    //
    // This example does NOT test that this certificate is chained to a Trusted Root Authority (or
revoked) on the server
    // and it allows for self signed certificates
    //

    if (certificate == null || !String.IsNullOrEmpty(errorString)) return false;

    // 1. Check time validity of certificate
    if (DateTime.Compare(DateTime.Now, certificate.NotBefore) < 0 || DateTime.Compare(DateTime.Now,
certificate.NotAfter) > 0) return false;

    // 2. Check subject name of certificate
    bool foundSubject = false;
    string[] certSubjectData = certificate.Subject.Split(new char[] { ',' },
StringSplitOptions.RemoveEmptyEntries);
    foreach (string s in certSubjectData)
    {
        if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
        {
            foundSubject = true;
            break;
        }
    }
    if (!foundSubject) return false;

    // 3. Check issuer name of certificate
    bool foundIssuerCN = false, foundIssuerO = false;
    string[] certIssuerData = certificate.Issuer.Split(new char[] { ',' },
StringSplitOptions.RemoveEmptyEntries);
    foreach (string s in certIssuerData)
    {
        if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
        {
            foundIssuerCN = true;
            if (foundIssuerO) break;
        }

        if (String.Compare(s.Trim(), "O=Microsoft Corp") == 0)
        {
            foundIssuerO = true;
            if (foundIssuerCN) break;
        }
    }
}

```

```
    if (!foundIssuerCN || !foundIssuerO) return false;

    // 4. Check thumbprint of certificate
    if (String.Compare(certificate.Thumbprint.Trim().ToUpper(),
"30757A2E831977D8BD9C8496E4C99AB26CB9622B") != 0) return false;

    // If you also want to test if the certificate chains to a Trusted Root Authority you can
uncomment the code below
    //
//X509Chain certChain = new X509Chain();
//certChain.Build(certificate);
//bool isValidCertChain = true;
//foreach (X509ChainElement chElement in certChain.ChainElements)
//{
//    if (!chElement.Certificate.Verify())
//    {
//        isValidCertChain = false;
//        break;
//    }
//}
//if (!isValidCertChain) return false;

    return true;
}
}
}
```

Scale up an app in Azure

9/25/2017 • 3 min to read • [Edit Online](#)

NOTE

The new **PremiumV2** tier gives you faster CPUs, SSD storage, and double the memory-to-core ratio than the existing pricing tiers. To scale up to **PremiumV2** tier, see [Configure PremiumV2 tier for App Service](#).

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 20 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 do not require you to change your code or redeploy your application.

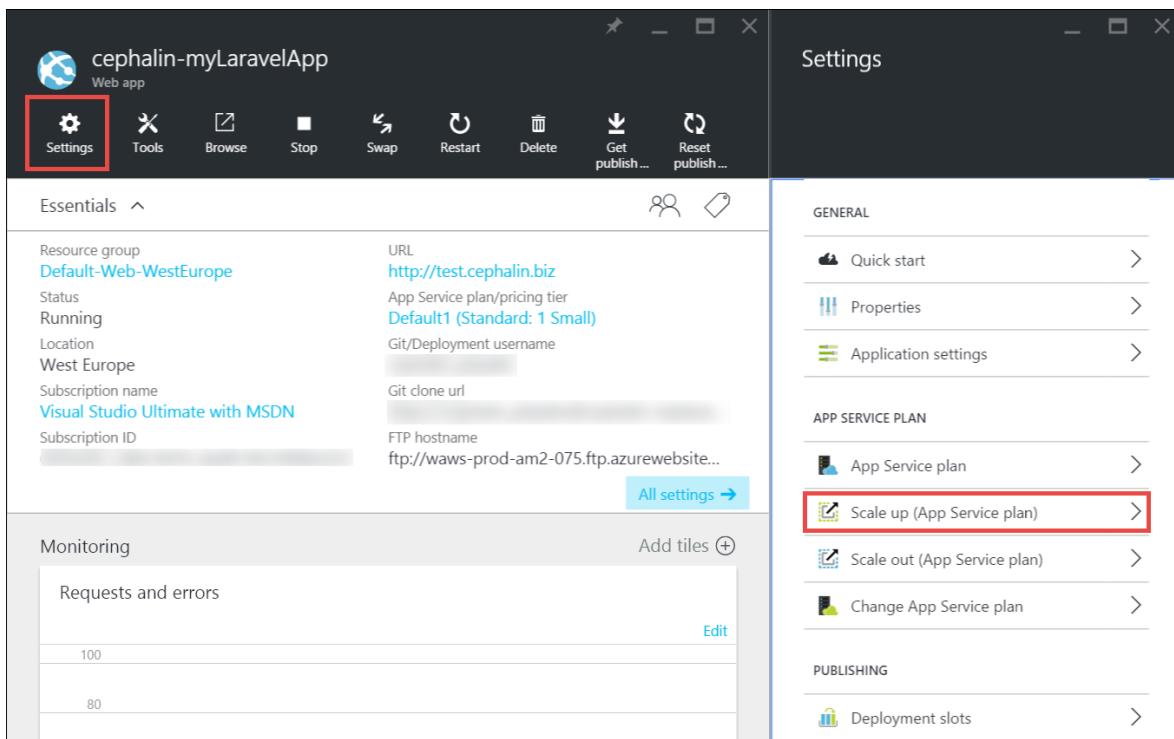
For information about the pricing and features of individual App Service plans, see [App Service Pricing Details](#).

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

1. In your browser, open the [Azure portal](#).
2. In your App Service app page, click **All settings**, and then click **Scale Up**.



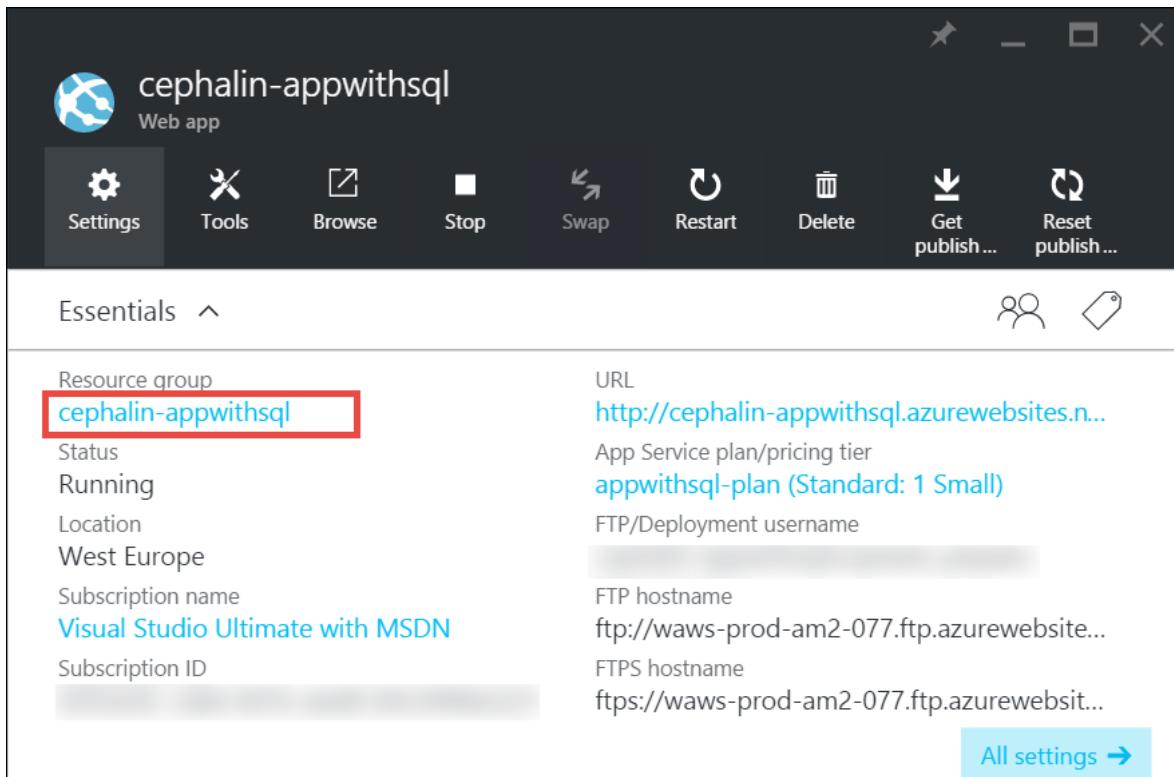
3. Choose your tier, and then click **Select**.

The **Notifications** tab will flash a green **SUCCESS** after the operation is complete.

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 are not managed by the App Service plan.

1. In **Essentials**, click the **Resource group** link.



2. In the **Summary** part of the **Resource group** page, click a resource that you want to scale. The following screenshot shows a SQL Database resource and an Azure Storage resource.

The screenshot shows the Azure portal interface for a resource group named 'cephalin-appwithsql'. At the top, there are navigation icons for Settings, Add, Columns, Delete, and Refresh. Below this is a section titled 'Essentials' with details about the subscription: name ('Visual Studio Ultimate with MSDN'), last deployment ('7/5/2016 (Succeeded)'), and location ('West US'). A blue button labeled 'All settings →' is present. A search bar at the bottom says 'Filter items...'. The main table lists resources by Name, Type, and Location:

NAME	TYPE	LOCATION
cephalin-appwithsql	Application Insights	Central US
cephalinappwithsqlserver	SQL server	North Europe
dbforapp	SQL database	North Europe
cephalinstorage4	Storage account	West Europe
appwithsql-plan	App Service plan	West Europe
cephalin-appwithsql	App Service	West Europe

3. For a SQL Database resource, click **Settings** > **Pricing tier** to scale the pricing tier.

The screenshot shows the Azure portal interface for a SQL database named 'dbforapp'. The left pane shows basic details: Resource group ('cephalin-appwithsql'), Status ('Online'), Location ('North Europe'), Subscription name ('Visual Studio Ultimate with MSDN'), and Subscription ID. The right pane is titled 'Settings' for 'dbforapp' and includes sections for Overview, Recommendations, Queries, Properties, Pricing tier (scale DTUs) (which is highlighted with a red box), Tags, and Security.

You can also turn on [geo-replication](#) for your SQL Database instance.

For an Azure Storage resource, click **Settings** > **Configuration** to scale up your storage options.

The screenshot shows the Azure Storage account settings for 'cephalinstorage4'. In the top left, there's a 'Settings' button which is highlighted with a red box. On the right side, under the 'GENERAL' section, the 'Configuration' item is also highlighted with a red box. The main pane displays various storage account details like resource group, status, location, and subscription information.

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](#).

NOTE

If you want to get started with Azure App Service before you sign up for an Azure account, go to [Try App Service](#) where you can immediately create a short-lived starter web app in App Service. No credit cards are required and there are no commitments.

Next steps

- For information about pricing, support, and SLA, visit the following links:

[Data Transfers Pricing Details](#)

[Microsoft Azure Support Plans](#)

[Service Level Agreements](#)

[SQL Database Pricing Details](#)

[Virtual Machine and Cloud Service Sizes for Microsoft Azure](#)

- For information about Azure App Service best practices, including building a scalable and resilient architecture, see [Best Practices: Azure App Service Web Apps](#).
- For videos about scaling App Service apps, see the following resources:
 - [When to Scale Azure Websites - with Stefan Schackow](#)
 - [Auto Scaling Azure Websites, CPU or Scheduled - with Stefan Schackow](#)
 - [How Azure Websites Scale - with Stefan Schackow](#)

Configure PremiumV2 tier for Azure App Service

9/25/2017 • 3 min to read • [Edit Online](#)

The new **PremiumV2** pricing tier provides [Dv2-series VMs](#) with faster processors, SSD storage, and double memory-to-core ratio compared to **Standard** tier. 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 a web app to **PremiumV2**, you need to have a Web App in Azure App Service that runs in a pricing tier lower than **PremiumV2**.

PremiumV2 availability

The PremiumV2 tier is currently available for App Service on *Windows* only. Linux containers are not yet supported.

PremiumV2 is already available in most Azure regions and growing. To see if it is available in your region, run the following Azure CLI command in the [Azure Cloud Shell](#):

```
az appservice list-locations --sku P1V2
```

If you receive an error during app creation or App Service plan creation, then **PremiumV2** is most likely not available for your region of choice.

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 Web App creation.

When configuring the App Service plan in the [Azure portal](#), select **Pricing tier**.

Choose one of the **PremiumV2** options and click **Select**.

New App Service Plan

Create a plan for the web app

* App Service plan	myAppServicePlan
* Location	West Europe
Pricing tier	S1 Standard

Choose your pricing tier

Browse the available plans and their features

I1 Isolated	I2 Isolated	I3 Isolated
1 Core	2 Core	4 Core
3.5 GB RAM	7 GB RAM	14 GB RAM
SSD and faster CPU Dv2 series workers	SSD and faster CPU Dv2 series workers	SSD and faster CPU Dv2 series workers
App Service Environment... Single tenant system	App Service Environment... Single tenant system	App Service Environment... Single tenant system
Runs in your vNET Network isolated	Runs in your vNET Network isolated	Runs in your vNET Network isolated
Private app access Using an ILB ASE	Private app access Using an ILB ASE	Private app access Using an ILB ASE
Used across ASE 1 TB Storage	Used across ASE 1 TB Storage	Used across ASE 1 TB Storage
Up to 100 instance(s) More upon request	Up to 100 instance(s) More upon request	Up to 100 instance(s) More upon request
223.20 USD/MONTH (PER INSTANCE)	446.40 USD/MONTH (PER INSTANCE)	892.80 USD/MONTH (PER INSTANCE)

P1V2 PremiumV2	P2V2 PremiumV2	P3V2 PremiumV2
1 Core	2 Core	4 Core
3.5 GB RAM	7 GB RAM	14 GB RAM
SSD and faster CPU Dv2 series workers	SSD and faster CPU Dv2 series workers	SSD and faster CPU Dv2 series workers
250 GB Storage	250 GB Storage	250 GB Storage
Custom domains / SSL SNI Incl & IP SSL Support	Custom domains / SSL SNI Incl & IP SSL Support	Custom domains / SSL SNI Incl & IP SSL Support
Up to 20 instance(s) * Subject to availability	Up to 20 instance(s) * Subject to availability	Up to 20 instance(s) * Subject to availability
20 slots Web app staging	20 slots Web app staging	20 slots Web app staging
Traffic Manager Geo availability	Traffic Manager Geo availability	Traffic Manager Geo availability
223.20 USD/MONTH (ESTIMATED)	446.40 USD/MONTH (ESTIMATED)	892.80 USD/MONTH (ESTIMATED)

P1 Premium

P2 Premium

P3 Premium

OK

Select

IMPORTANT

If you do not see **P1V2**, **P2V2**, and **P3V2** as options, either **PremiumV2** is not available in your region of choice, or you are configuring a Linux App Service plan, which does not support **PremiumV2**.

Scale up an existing app to PremiumV2 tier

Before scaling an existing app to **PremiumV2** tier, make sure that **PremiumV2** is available in your region. For information, see [PremiumV2 availability](#). If it is not available in your region, see [Scale up from an unsupported region](#).

Depending on your hosting environment, scaling up may require extra steps.

In the [Azure portal](#), open your App Service app page.

In the left navigation of your App Service app page, select **Scale up (App Service plan)**.

The screenshot shows the Azure portal interface for an App Service named 'app-scaling-premiumv2'. On the left, there's a sidebar with various options 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 app details: Resource group (myResourceGroupWin), Status (Running), Location (West Europe), Subscription (change), and Subscription ID. Below this is a preview section showing 'Http 5xx' and some data.

Select one of the **PremiumV2** sizes, then click **Select**.

This screenshot shows the 'Choose your pricing tier' dialog. It lists three PremiumV2 plans: P1V2 PremiumV2, P2V2 PremiumV2, and P3V2 PremiumV2. Each plan includes details like Core count, RAM, storage, and features. The P1V2 plan is highlighted with a red box. At the bottom is a 'Select' button.

P1V2 PremiumV2	P2V2 PremiumV2	P3V2 PremiumV2
1 Core	2 Core	4 Core
3.5 GB RAM	7 GB RAM	14 GB RAM
SSD and faster CPU Dv2 series workers	SSD and faster CPU Dv2 series workers	SSD and faster CPU Dv2 series workers
250 GB Storage	250 GB Storage	250 GB Storage
Custom domains / SSL SNI Incl & IP SSL Support	Custom domains / SSL SNI Incl & IP SSL Support	Custom domains / SSL SNI Incl & IP SSL Support
<input checked="" type="checkbox"/> Up to 20 instance(s) * Subject to availability	<input checked="" type="checkbox"/> Up to 20 instance(s) * Subject to availability	<input checked="" type="checkbox"/> Up to 20 instance(s) * Subject to availability
20 slots Web app staging	20 slots Web app staging	20 slots Web app staging
Traffic Manager Geo availability	Traffic Manager Geo availability	Traffic Manager Geo availability
223.20 USD/MONTH (ESTIMATED)	446.40 USD/MONTH (ESTIMATED)	892.80 USD/MONTH (ESTIMATED)

If your operation finishes successfully, your app's overview page shows that it is now in a **PremiumV2** tier.

The screenshot shows the app's overview page. It displays basic information: Resource group (myResourceGroupWin), Status (Running), Location (West Europe), Subscription (change), and Subscription ID. It also shows the URL (<http://app-scaling-premiumv2.azurewebsites.net>) and the selected App Service plan/pricing tier, which is 'app-scaling-premiumv2 (PremiumV2: 1 Small)', highlighted with a red box. Other connection details like FTP/username, hostname, and SSL hostnames are also listed.

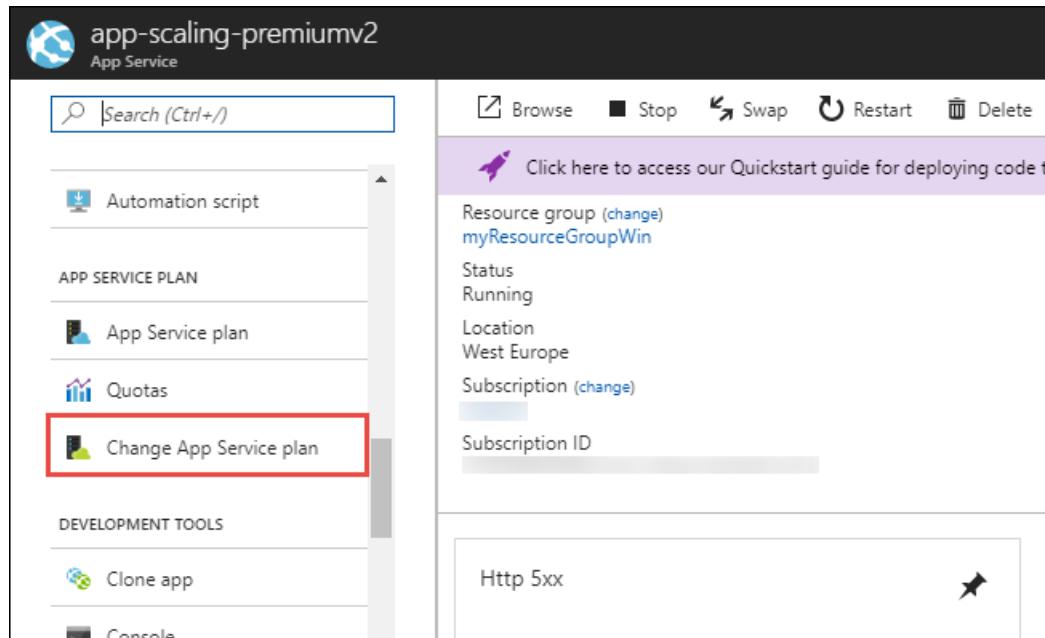
If you get an error

Some App Service plans cannot scale up to the PremiumV2 tier. If your scale-up operation gives you an error, you need a new App Service plan for your app.

Create a Windows App Service plan in the same region and resource group as your existing App Service app.

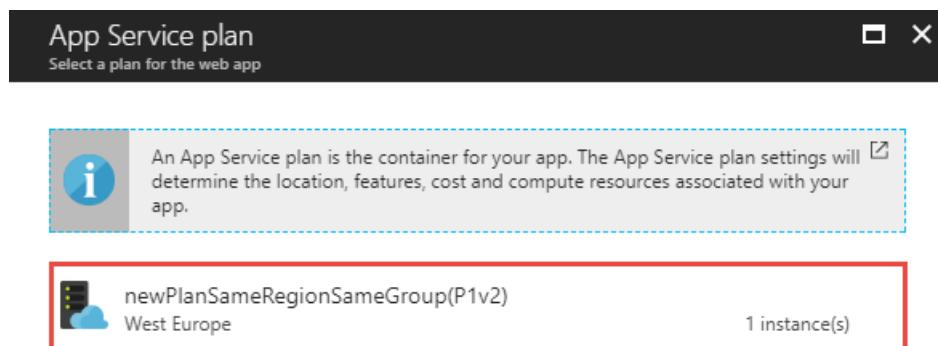
Follow the steps at [Create an app in PremiumV2 tier](#) to set it to **PremiumV2** tier. If desired, use the same scale-out configuration as your existing App Service plan (number of instances, autoscale, and so on).

Open your App Service app page again. In the left navigation of your App Service, select **Change App Service plan**.



The screenshot shows the Azure portal interface for an App Service named 'app-scaling-premiumv2'. On the left, there's a sidebar with various options like 'Automation script', 'APP SERVICE PLAN', 'App Service plan' (which is selected and highlighted with a red box), 'Quotas', and 'Change App Service plan' (also highlighted with a red box). The main content area displays details about the current service plan: 'Resource group (change) myResourceGroupWin', 'Status Running', 'Location West Europe', 'Subscription (change)', and 'Subscription ID'. At the bottom, there's a status message 'Http 5xx' with a refresh icon.

Select the App Service plan you just created.



The screenshot shows a modal dialog titled 'App Service plan' with the sub-instruction 'Select a plan for the web app'. Inside, there's an information icon with a tooltip explaining what an App Service plan is. Below that, a list shows a single plan: 'newPlanSameRegionSameGroup(P1v2)' located 'West Europe' with '1 instance(s)'. This last item is also highlighted with a red box.

Once the change operation completes, your app is running in **PremiumV2** tier.

Scale up from an unsupported region

If your app runs in a region where **PremiumV2** is not yet available, you can move your app to a different region to take advantage of **PremiumV2**. You have two options:

- Create an app in new **PremiumV2** plan, then redeploy your application code. Follow the steps at [Create an app in PremiumV2 tier](#) to set it to **PremiumV2** tier. If desired, use the same scale-out configuration as your existing App Service plan (number of instances, autoscale, and so on).
- 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.

The screenshot shows the Azure portal interface for an App Service named 'app-scaling-premiumv2'. In the top left, there's a search bar labeled 'Search (Ctrl+ /)'. Below it, a sidebar titled 'DEVELOPMENT TOOLS' contains several options: 'Clone app' (which is highlighted with a red box), 'Console', 'Advanced Tools', 'App Service Editor (Preview)', and 'Performance test'. On the right, there's a summary card with details: Resource group (myResourceGroupWin), Status (Running), Location (West Europe), Subscription (antps31), and Subscription ID (7c1b7bab-00b2-4cb7-924e-205c4f411810). At the top, there are buttons for 'Browse', 'Stop', 'Swap', 'Restart', and 'Delete'.

In the **Clone app** page, you can create a new App Service plan in the region you want, and specify the settings 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

The following command creates an App Service plan in *P1V2*. The options for `-WorkerSize` are *Small*, *Medium*, and *Large*.

```
New-AzureRmAppServicePlan -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](#)

Scale instance count manually or automatically

12/13/2017 • 6 min to read • [Edit Online](#)

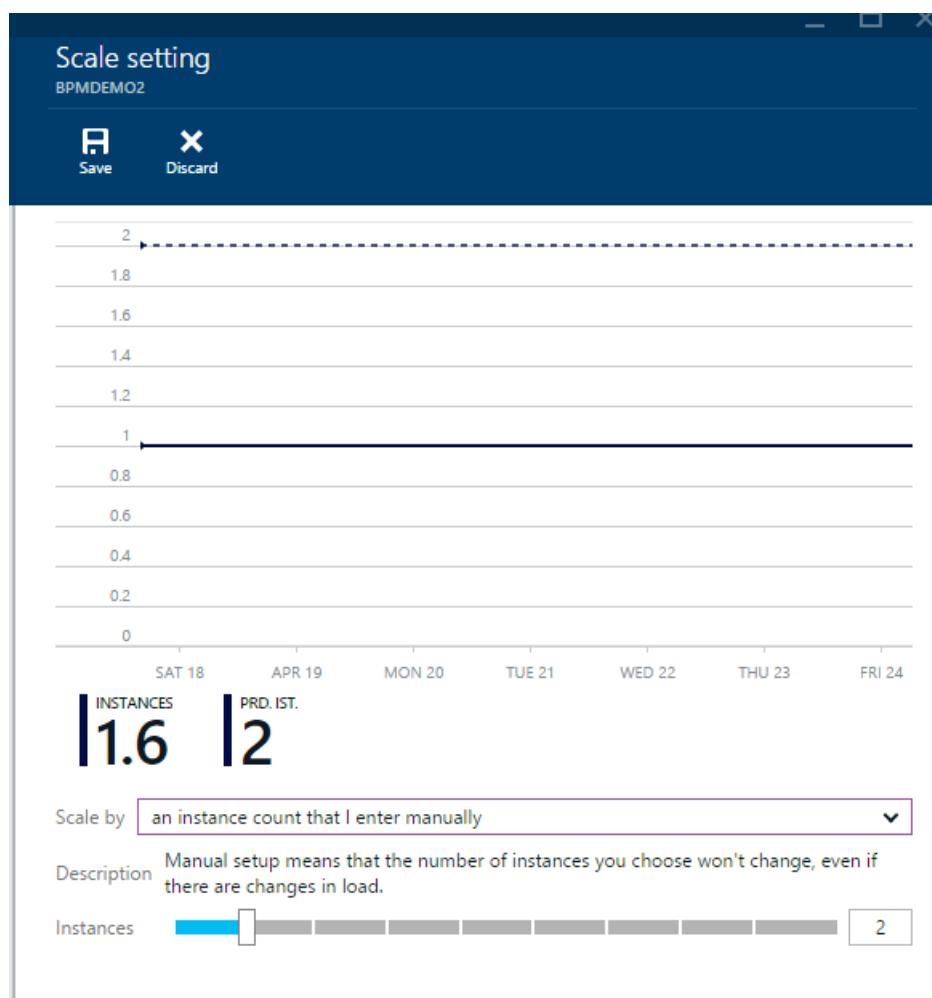
In the [Azure Portal](#), you can manually set the instance count of your service, or, you can set parameters to have it automatically scale based on demand. This is typically referred to as *Scale out* or *Scale in*.

Before scaling based on instance count, you should consider that scaling is affected by **Pricing tier** in addition to instance count. Different pricing tiers can have different numbers cores and memory, and so they will have better performance for the same number of instances (which is *Scale up* or *Scale down*). This article specifically covers *Scale in* and *out*.

You can scale in the portal, and you can also use the [REST API](#) or [.NET SDK](#) to adjust scale manually or automatically.

Scaling manually

1. In the [Azure Portal](#), click **Browse**, then navigate to the resource you want to scale, such as an **App Service plan**.
2. Click **Settings > Scale out (App Service plan)**.
3. At the top of the **Scale** blade you can see a history of autoscale actions of the service.



NOTE

Only actions that are performed by autoscale will show up in this chart. If you manually adjust the instance count, the change will not be reflected in this chart.

4. You can manually adjust the number **Instances** with slider.
5. Click the **Save** command and you'll be scaled to that number of instances almost immediately.

Scaling based on a pre-set metric

If you want the number of instances to automatically adjust based on a metric, select the metric you want in the **Scale by** dropdown. For example, for an **App Service plan** you can scale by **CPU Percentage**.

1. When you select a metric you'll get a slider, and/or, text boxes to enter the number of instances you want to scale between:



Autoscale will never take your service below or above the boundaries that you set, no matter your load.

2. Second, you choose the target range for the metric. For example, if you chose **CPU percentage**, you can set a target for the average CPU across all of the instances in your service. A scale out will happen when the average CPU exceeds the maximum you define, likewise, a scale in will happen whenever the average CPU drops below the minimum.
3. Click the **Save** command. Autoscale will check every few minutes to make sure that you are in the instance range and target for your metric. When your service receives additional traffic, you will get more instances without doing anything.

Scale based on other metrics

You can scale based on metrics other than the presets that appear in the **Scale by** dropdown, and can even have a complex set of scale out and scale in rules.

Adding or changing a rule

1. Choose the **schedule and performance rules** in the **Scale by** dropdown:

Scale by **schedule and performance rules**

Description Create your own set of rules. Create a schedule that adjusts your instance counts based on time and performance metrics.

Default, scale 2 - 10

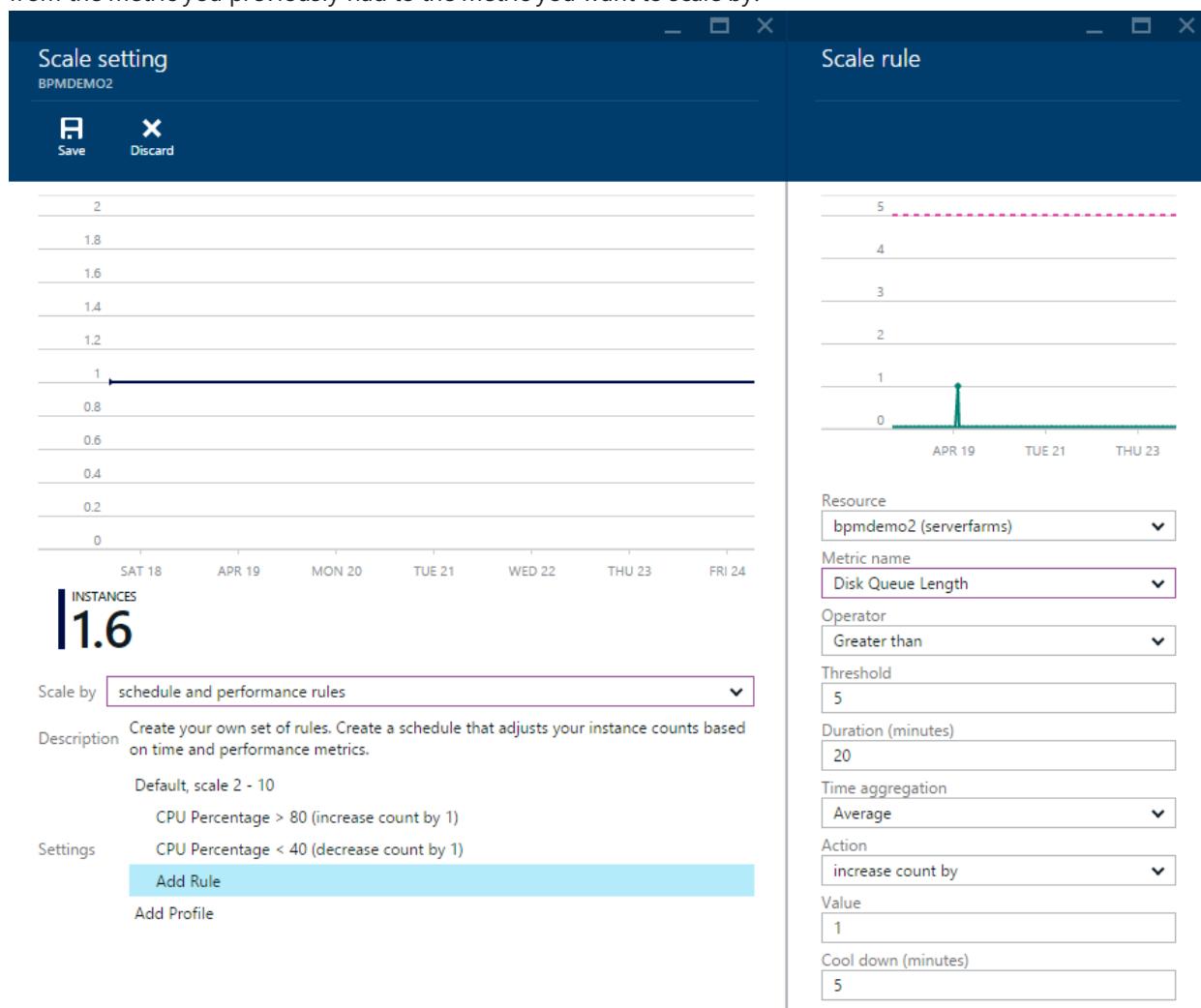
CPU Percentage > 80 (increase count by 1)

Settings CPU Percentage < 40 (decrease count by 1)

Add Rule

Add Profile

2. If you previously had autoscale, on you'll see a view of the exact rules that you had.
3. To scale based on another metric click the **Add Rule** row. You can also click one of the existing rows to change from the metric you previously had to the metric you want to scale by.



4. Now you need to select which metric you want to scale by. When choosing a metric there are a couple things to consider:

- The *resource* the metric comes from. Typically, this will be the same as the resource you are scaling. However, if you want to scale by the depth of a Storage queue, the resource is the queue that you want to scale by.
- The *metric name* itself.

- The *time aggregation* of the metric. This is how the data is combine over the *duration*.
- After choosing your metric you choose the threshold for the metric, and the operator. For example, you could say **Greater than 80%**.
 - Then choose the action that you want to take. There are a couple different type of actions:
 - Increase or decrease by - this will add or remove the **Value** number of instances you define
 - Increase or decrease percent - this will change the instance count by a percent. For example, you could put 25 in the **Value** field, and if you currently had 8 instances, 2 would be added.
 - Increase or decrease to - this will set the instance count to the **Value** you define.
 - Finally, you can choose cool down - how long this rule should wait after the previous scale action to scale again.
 - After configuring your rule hit **OK**.
 - Once you have configured all of the rules you want, be sure to hit the **Save** command.

Scaling with multiple steps

The examples above are pretty basic. However, if you want to be more agressive about scaling up (or down), you can even add multiple scale rules for the same metric. For example, you can define two scale rules on CPU percentage:

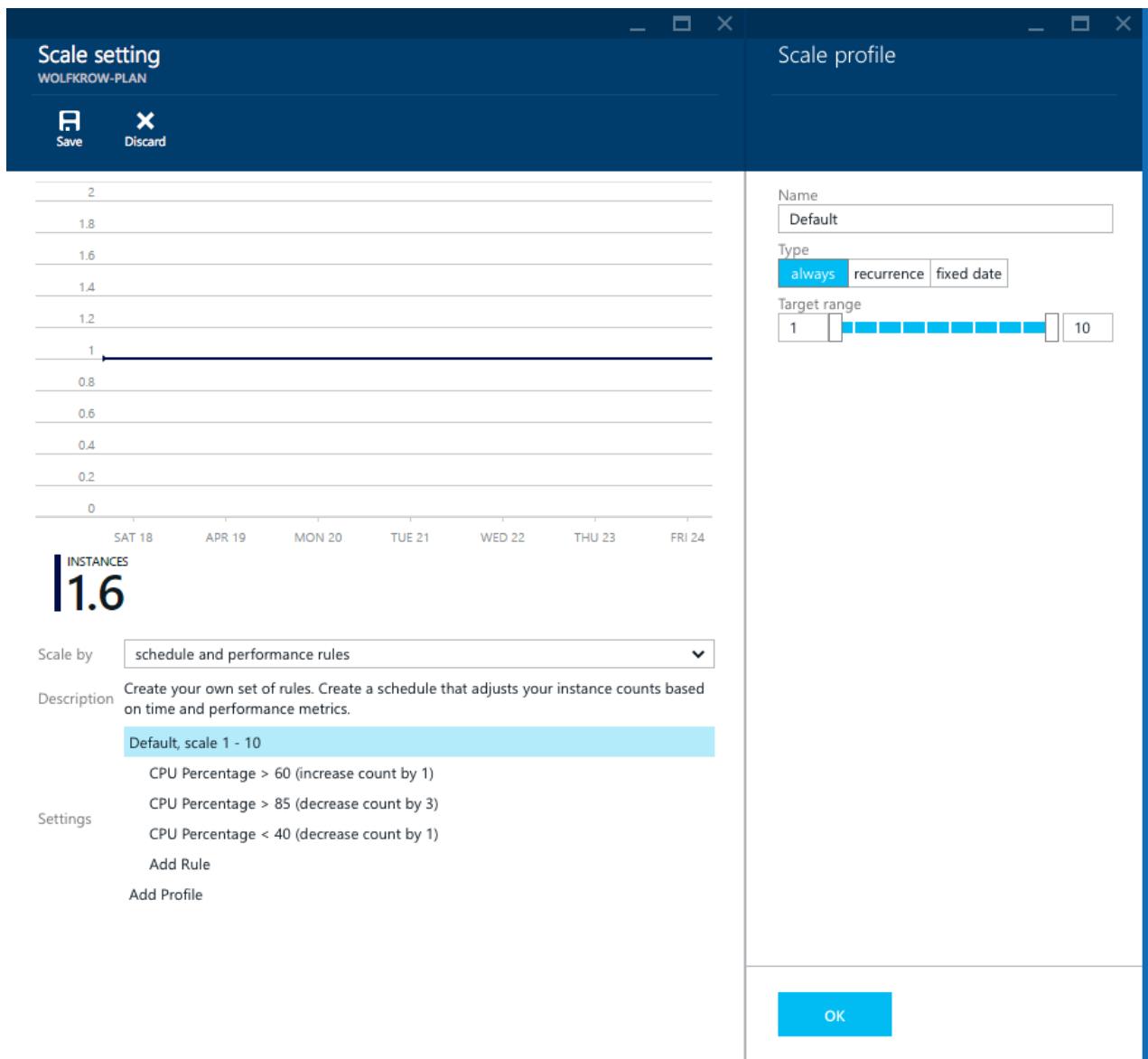
- Scale out by 1 instance if CPU percentage is above 60%
- Scale out by 3 instances if CPU percentage is above 85%

Default, scale 1 - 10
 Settings
 CPU Percentage > 60 (increase count by 1)
 CPU Percentage > 85 (decrease count by 3)
 CPU Percentage < 40 (decrease count by 1)
 Add Rule
 Add Profile

With this additional rule, if your load exceeds 85% before a scale action, you will get two additional instances instead of one.

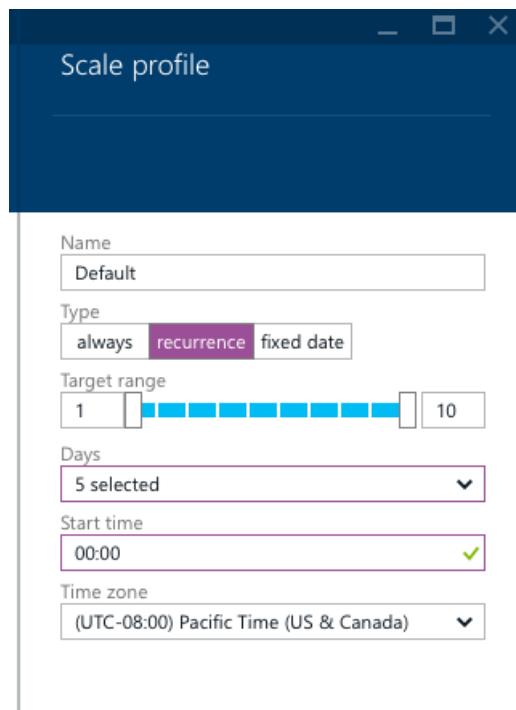
Scale based on a schedule

By default, when you create a scale rule it will always apply. You can see that when you click on the profile header:



However, you may want to have more aggressive scaling during the day, or the week, than on the weekend. You could even shut down your service entirely off working hours.

1. To do this, on the profile you have, select **recurrence** instead of **always**, and choose the times that you want the profile to apply.
2. For example, to have a profile that applies during the week, in the **Days** dropdown uncheck **Saturday** and **Sunday**.
3. To have a profile that applies during the daytime, set the **Start time** to the time of day that you want to start at.



4. Click **OK**.
5. Next, you will need to add the profile that you want to apply at other times. Click the **Add Profile** row.

6. Name your new, second, profile, for example you could call it **Off work**.
7. Then select **recurrence** again, and choose the instance count range you want during this time.
8. As with the Default profile, choose the **Days** you want this profile to apply to, and the **Start time** during the

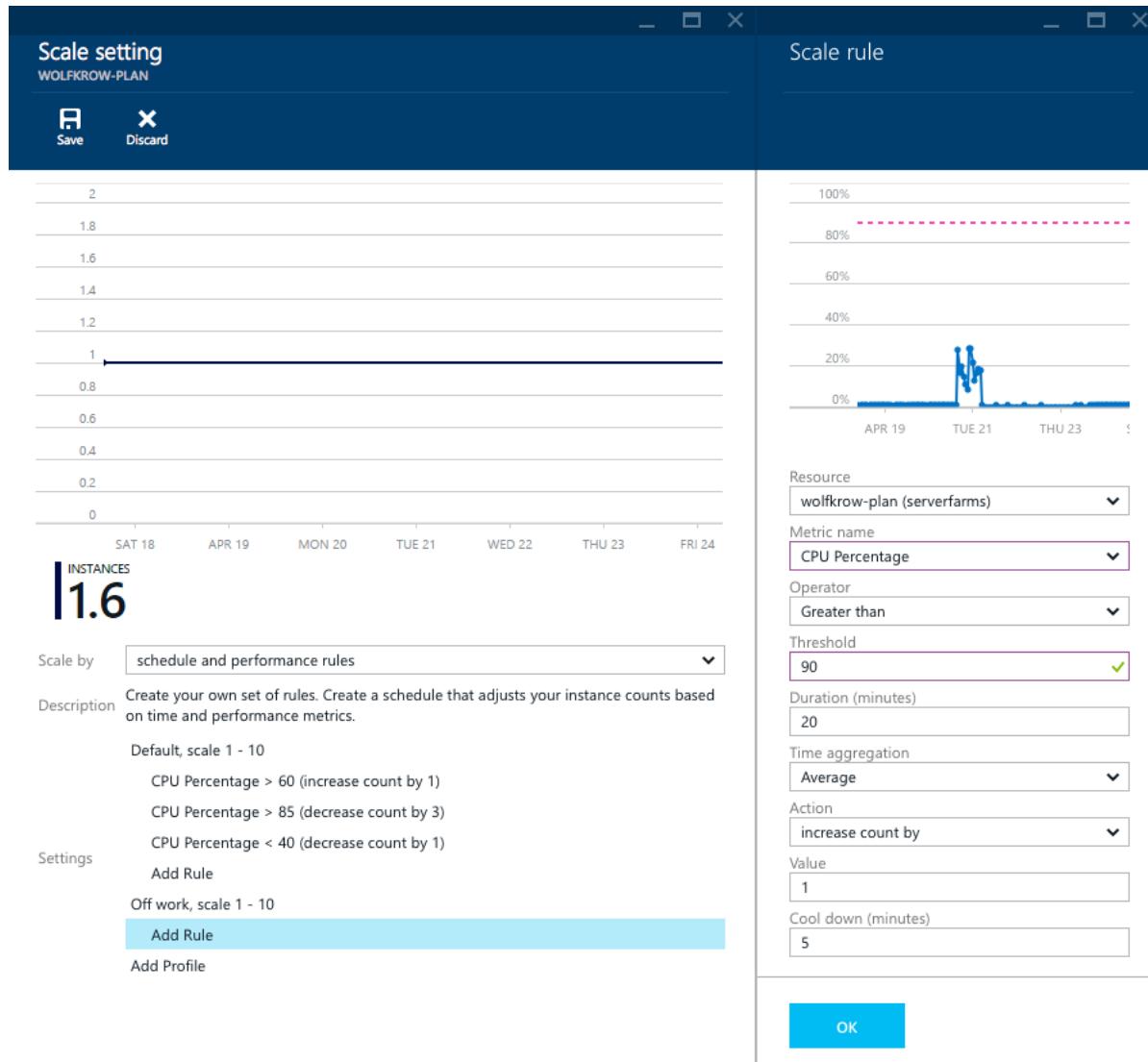
day.

NOTE

Autoscale will use the Daylight savings rules for whichever **Time zone** you select. However, during Daylight savings time the UTC offset will show the base Time zone offset, not the Daylight savings UTC offset.

9. Click **OK**.

10. Now, you will need to add whatever rules you want to apply during your second profile. Click **Add Rule**, and then you could construct the same rule you have during the Default profile.



11. Be sure to create both a rule for scale out and scale in, or else during the profile the instance count will only grow (or decrease).

12. Finally, click **Save**.

Next steps

- [Monitor service metrics](#) to make sure your service is available and responsive.
- [Enable monitoring and diagnostics](#) to collect detailed high-frequency metrics on your service.
- [Receive alert notifications](#) whenever operational events happen or metrics cross a threshold.
- [Monitor application performance](#) if you want to understand exactly how your code is performing in the cloud.
- [View events and activity log](#) to learn everything that has happened in your service.
- [Monitor availability and responsiveness of any web page](#) with Application Insights so you can find out if your

page is down.

How to: Monitor Apps in Azure App Service

1/2/2018 • 5 min to read • [Edit Online](#)

App Service provides built in monitoring functionality in the [Azure portal](#). The Azure portal includes the ability to review **quotas** and **metrics** for an app as well as the App Service plan, setting up **alerts** and even **scaling** automatically based on these metrics.

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

Understanding Quotas and Metrics

Quotas

Applications hosted in App Service are subject to certain *limits* on the resources they can use. The limits are defined by the **App Service plan** associated with the app.

NOTE

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

If the application is hosted in a **Free** or **Shared** plan, then the limits on the resources the app can use are defined by **Quotas**.

If the application is hosted in a **Basic**, **Standard** or **Premium** plan, then the limits on the resources they can use are set by the **size** (Small, Medium, Large) and **instance count** (1, 2, 3, ...) of the **App Service plan**.

Quotas for Free or Shared apps are:

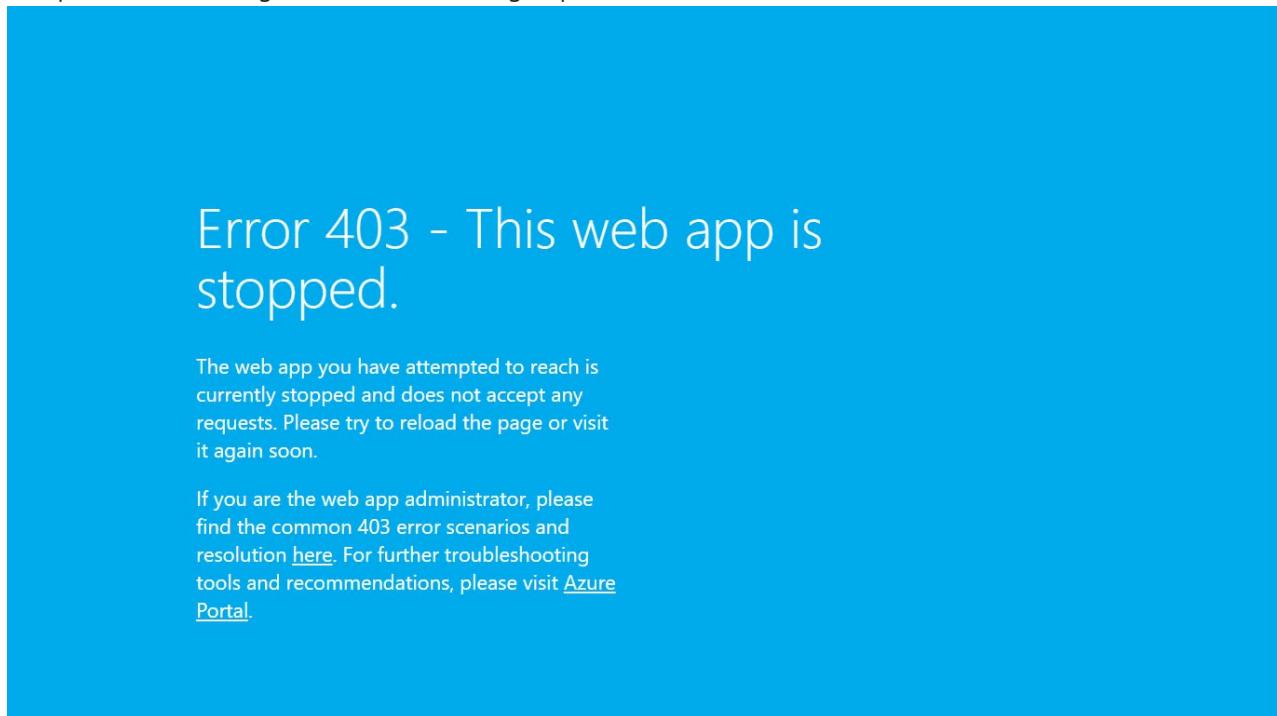
- **CPU(Short)**
 - Amount of CPU allowed for this application in a 5-minute interval. This quota resets every five minutes.
- **CPU(Day)**
 - Total amount of CPU allowed for this application in a day. This quota resets every 24 hours at midnight UTC.
- **Memory**
 - Total amount of memory allowed for this application.
- **Bandwidth**
 - Total amount of outgoing bandwidth allowed for this application in a day. This quota resets every 24 hours at midnight UTC.
- **Filesystem**
 - Total amount of storage allowed.

The only quota applicable to apps hosted on **Basic**, **Standard**, and **Premium** plans is **Filesystem**.

More information about the specific quotas, limits, and features available to the different App Service SKUs can be found here: [Azure Subscription Service Limits](#)

Quota Enforcement

If an application exceeds the **CPU (short)**, **CPU (Day)**, or **bandwidth** quota then the application is stopped until the quota resets. During this time, all incoming requests result in an **HTTP 403**.



If the application **memory** quota is exceeded, then the application is restarted.

If the **Filesystem** quota is exceeded, then any write operation fails, which includes any writes to logs.

Quotas can be increased or removed from your app by upgrading your App Service plan.

Metrics

Metrics provide information about the app, or App Service plan's behavior.

For an **Application**, the available metrics are:

- **Average Response Time**
 - The average time taken for the app to serve requests in ms.
- **Average memory working set**
 - The average amount of memory in MiBs used by the app.
- **CPU Time**
 - The amount of CPU in seconds consumed by the app. For more information about this metric, see: [CPU time vs CPU percentage](#)
- **Data In**
 - The amount of incoming bandwidth consumed by the app in MiBs.
- **Data Out**
 - The amount of outgoing bandwidth consumed by the app in MiBs.
- **Http 2xx**
 - Count of requests resulting in an HTTP status code ≥ 200 but < 300 .
- **Http 3xx**
 - Count of requests resulting in an HTTP status code ≥ 300 but < 400 .
- **Http 401**
 - Count of requests resulting in HTTP 401 status code.
- **Http 403**
 - Count of requests resulting in HTTP 403 status code.
- **Http 404**

- Count of requests resulting in HTTP 404 status code.
- **Http 406**
 - Count of requests resulting in HTTP 406 status code.
- **Http 4xx**
 - Count of requests resulting in an HTTP status code ≥ 400 but < 500 .
- **Http Server Errors**
 - Count of requests resulting in an HTTP status code ≥ 500 but < 600 .
- **Memory working set**
 - Current amount of memory used by the app in MiBs.
- **Requests**
 - Total number of requests regardless of their resulting HTTP status code.

For an **App Service plan**, the available metrics are:

NOTE

App Service plan metrics are only available for plans in **Basic**, **Standard**, and **Premium** tiers.

- **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 application that might be slowing down due to 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** and **CPU percentage**

CPU Time is useful for apps hosted in **Free** or **Shared** plans since one of their quotas is defined in CPU minutes used by the app.

CPU percentage is useful for apps hosted in **basic**, **standard**, and **premium** plans since they can be scaled out. CPU percentage is a good indication of the overall usage across all instances.

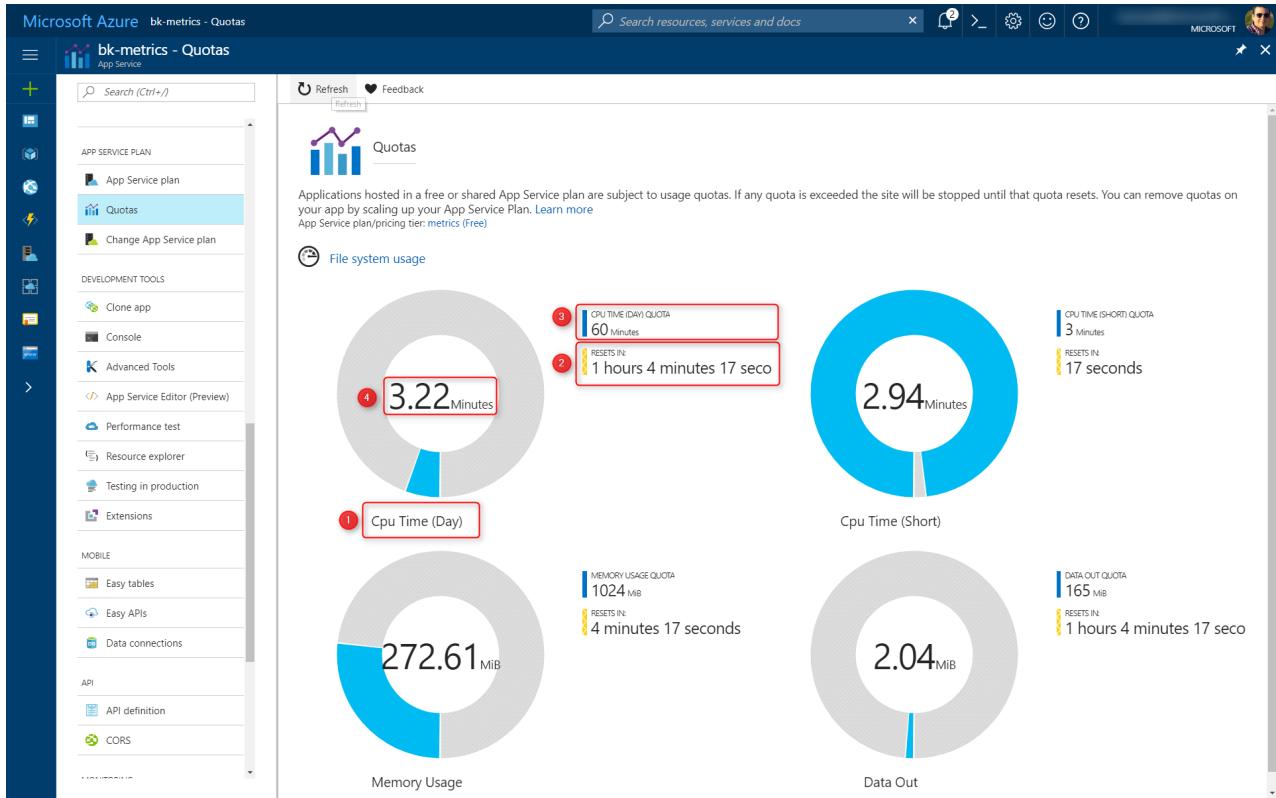
Metrics Granularity and Retention Policy

Metrics for an application and app service plan are logged and aggregated by the service with the following granularities and retention policies:

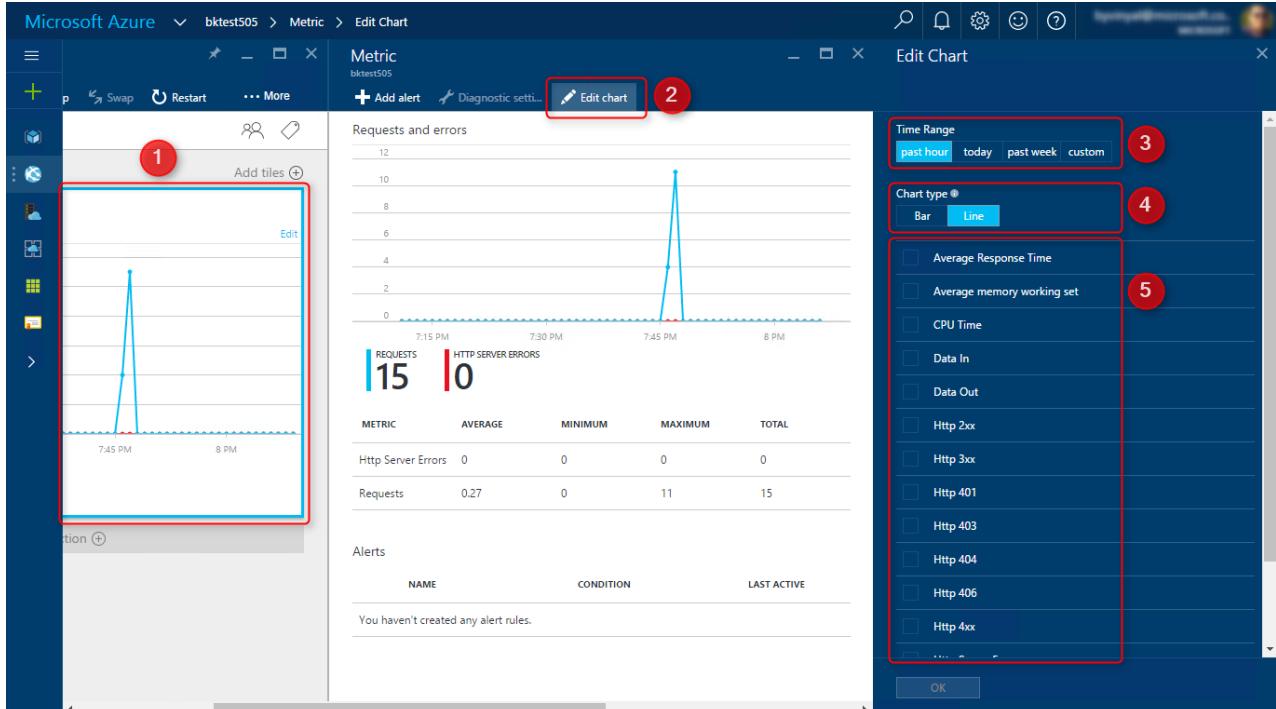
- **Minute** granularity metrics are retained for **30 hours**
- **Hour** granularity metrics are retained for **30 days**
- **Day** granularity metrics are retained for **30 days**

Monitoring Quotas and Metrics in the Azure portal.

You can review the status of the different **quotas** and **metrics** affecting an application in the [Azure portal](#).



Quotas can be found under Settings>**Quotas**. The UX allows you to review: (1) the quotas name, (2) its reset interval, (3) its current limit, and (4) current value.



Metrics can be accessed directly from the resource page. You can also customize the chart by: (1) click on it, and select (2) **edit chart**. From here you can change the (3) **time range**, (4) **chart type**, and (5) **metrics** to display.

You can learn more about metrics here: [Monitor service metrics](#).

Alerts and Autoscale

Metrics for an App or App Service plan can be hooked up to alerts. To learn more about it, see [Receive alert](#)

notifications.

App Service apps hosted in basic, standard, or premium App Service plans support **autoscale**. Autoscale allows you to configure rules that monitor the App Service plan metrics. Rules can increase or decrease the instance count providing additional resources as needed. Rules can also help you save money when the application is over-provisioned. You can learn more about auto scale here: [How to Scale](#) and here [Best practices for Azure Monitor autoscaling](#)

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Enable diagnostics logging for web apps in Azure App Service

12/6/2017 • 12 min to read • [Edit Online](#)

Overview

Azure provides built-in diagnostics to assist with debugging an [App Service web 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](#), Azure PowerShell, and the Azure Command-Line Interface (Azure CLI) to work with diagnostic logs. For information on working with diagnostic logs using Visual Studio, see [Troubleshooting Azure in Visual Studio](#).

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

Web server diagnostics and application diagnostics

App Service web apps provide diagnostic functionality for logging information from both the web server and the web application. These are logically separated into **web server diagnostics** and **application diagnostics**.

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). It 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. It is useful if you are attempting to increase site performance or isolate what is causing a specific HTTP error to be returned.
- **Web Server Logging** - Information about HTTP transactions using the [W3C extended log file format](#). It is useful when determining overall site metrics such as the number of requests handled or how many requests are from a specific IP address.

Application diagnostics

Application diagnostics allows 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. For example:

```
System.Diagnostics.Trace.TraceError("If you're seeing this, something bad happened");
```

At runtime, you can retrieve these logs to help with troubleshooting. For more information, see [Troubleshooting Azure web apps in Visual Studio](#).

App Service web apps also log deployment information when you publish content to a web app. It happens automatically and there are no configuration settings for deployment logging. Deployment logging allows you to determine why a deployment failed. For example, if you are using a custom deployment script, you might use deployment logging to determine why the script is failing.

How to enable diagnostics

To enable diagnostics in the [Azure portal](#), go to the page for your web app and click **Settings > Diagnostics logs**.

The screenshot shows two side-by-side windows from the Azure portal.

Left Window (Settings):

- Header: Settings
- Search bar: Search settings
- SUPPORT & TROUBLESHOOTING:
 - Check health
 - Troubleshoot
 - New support request
- GENERAL:
 - Quick start
 - Properties
 - Application settings
- APP SERVICE PLAN:
 - App Service Plan
 - Scale Up (App Service Plan)
 - Scale Out (App Service Plan)
- FEATURES:
 - Backups
 - Authentication / Authorization
 - Diagnostics logs
- PUBLISHING

Right Window (Logs):

- Header: Logs (logging1234)
- Buttons: Save, Discard
- Application Logging (Filesystem) **On**
- Application Logging (Blob) **Off**
- Web server logging **Storage**
- Detailed error messages **On**
- Failed request tracing **Off**
- Download logs
 - FTP/deployment user name: logging1234\steyer
 - FTP: ftp://waws-prod-bay-037.ftp.azurewebsites.net
 - FTPS: ftps://waws-prod-bay-037.ftp.azurewebsites.net

When you enable **application diagnostics**, you also choose the **Level**. This setting allows you to filter the information captured to **informational**, **warning**, or **error** information. Setting it to **verbose** logs all information produced by the application.

NOTE

Unlike changing the web.config file, enabling Application diagnostics or changing diagnostic log levels does not recycle the app domain that the application runs within.

For **Application logging**, you can turn on the file system option temporarily for debugging purposes. This option turns off automatically in 12 hours. You can also turn on the blob storage option to select a blob container to write logs to.

For **Web server logging**, you can select **storage** or **file system**. Selecting **storage** allows you to select a storage

account, and then a blob container that the logs are written to.

If you store logs on the file system, the files can be accessed by FTP, or downloaded as a Zip archive by using the Azure PowerShell or Azure Command-Line Interface (Azure CLI).

By default, logs are not automatically deleted (with the exception of **Application Logging (Filesystem)**). To automatically delete logs, set the **Retention Period (Days)** field.

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 or table again. Save your setting.

Any combination of file system, table storage, or blob storage can be enabled at the same time, and have individual log level configurations. For example, you may wish to log errors and warnings to blob storage as a long-term logging solution, while enabling file system logging with a level of verbose.

While all three storage locations provide the same basic information for logged events, **table storage** and **blob storage** log additional information such as the instance ID, thread ID, and a more granular timestamp (tick format) than logging to **file system**.

NOTE

Information stored in **table storage** or **blob storage** can only be accessed using a storage client or an application that can directly work with these storage systems. For example, Visual Studio 2013 contains a Storage Explorer that can be used to explore table or blob storage, and HDInsight can access data stored in blob storage. You can also write an application that accesses Azure Storage by using one of the [Azure SDKs](#).

NOTE

Diagnostics can also be enabled from Azure PowerShell using the **Set-AzureWebsite** cmdlet. If you have not installed Azure PowerShell, or have not configured it to use your Azure Subscription, see [How to Use Azure PowerShell](#).

How to: Download logs

Diagnostic information stored to the web app file system can be accessed directly using FTP. It can also be downloaded as a Zip archive using Azure PowerShell or the Azure Command-Line Interface.

The directory structure that the logs are stored in is as follows:

- **Application logs** - /LogFiles/Application/. This folder contains one or more text files containing information produced by application logging.
- **Failed Request Traces** - /LogFiles/W3SVC#####/. This folder contains an XSL file and one or more XML files. Ensure that you download the XSL file into the same directory as the XML file(s) because the XSL file provides functionality for formatting and filtering the contents of the XML file(s) when viewed in Internet Explorer.
- **Detailed Error Logs** - /LogFiles/DetailedErrors/. This folder contains one or more .htm files that provide extensive information for any HTTP errors that have occurred.
- **Web Server Logs** - /LogFiles/http/RawLogs. This folder contains one or more text files formatted using the [W3C extended log file format](#).
- **Deployment logs** - /LogFiles/Git. This folder contains logs generated by the internal deployment processes

used by Azure web apps, as well as logs for Git deployments.

FTP

To open an FTP connection to your app's FTP server, see [Deploy your app to Azure App Service using FTP/S](#).

Once connected to your web app's FTP/S server, open the **LogFiles** folder, where the log files are stored.

Download with Azure PowerShell

To download the log files, start a new instance of Azure PowerShell and use the following command:

```
Save-AzureWebSiteLog -Name webappname
```

This command saves the logs for the web app specified by the **-Name** parameter to a file named **logs.zip** in the current directory.

NOTE

If you have not installed Azure PowerShell, or have not configured it to use your Azure Subscription, see [How to Use Azure PowerShell](#).

Download with Azure Command-Line Interface

To download the log files using the Azure Command Line Interface, open a new command prompt, PowerShell, Bash, or Terminal session and enter the following command:

```
azure site log download webappname
```

This command saves the logs for the web app named 'webappname' to a file named **diagnostics.zip** in the current directory.

NOTE

If you have not installed the Azure Command-Line Interface (Azure CLI), or have not configured it to use your Azure Subscription, see [How to Use Azure CLI](#).

How to: View logs in Application Insights

Visual Studio Application Insights provides tools for filtering and searching logs, and for correlating the logs with requests and other events.

1. Add the Application Insights SDK to your project in Visual Studio.
 - In Solution Explorer, right-click your project and choose Add Application Insights. The interface guides you through steps that include creating an Application Insights resource. [Learn more](#)
2. Add the Trace Listener package to your project.
 - Right-click your project and choose Manage NuGet Packages. Select **Microsoft.ApplicationInsights.TraceListener** [Learn more](#)
3. Upload your project and run it to generate log data.
4. In the [Azure portal](#), browse to your new Application Insights resource, and open **Search**. You should see your log data, along with request, usage, and other telemetry. Some telemetry might take a few minutes to arrive: click Refresh. [Learn more](#)

[Learn more about performance tracking with Application Insights](#)

How to: Stream logs

While developing an application, it is often useful to see logging information in near-real time. You can stream logging information to your development environment using either Azure PowerShell or the Azure Command-Line Interface.

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.

NOTE

Log streaming also streams information written to any text file stored in the **D:\home\LogFiles** folder.

Streaming with Azure PowerShell

To stream logging information, start a new instance of Azure PowerShell and use the following command:

```
Get-AzureWebSiteLog -Name webappname -Tail
```

This command connects to the web app specified by the **-Name** parameter and begin streaming information to the PowerShell window as log events occur on the web app. Any information written to files ending in .txt, .log, or .htm that are stored in the /LogFiles directory (d:/home/logfiles) is streamed to the local console.

To filter specific events, such as errors, use the **-Message** parameter. For example:

```
Get-AzureWebSiteLog -Name webappname -Tail -Message Error
```

To filter specific log types, such as HTTP, use the **-Path** parameter. For example:

```
Get-AzureWebSiteLog -Name webappname -Tail -Path http
```

To see a list of available paths, use the **-ListPath** parameter.

NOTE

If you have not installed Azure PowerShell, or have not configured it to use your Azure Subscription, see [How to Use Azure PowerShell](#).

Streaming with Azure Command-Line Interface

To stream logging information, open a new command prompt, PowerShell, Bash, or Terminal session and enter the following command:

```
az webapp log tail --name webappname --resource-group myResourceGroup
```

This command connects to the web app named 'webappname' and begin streaming information to the window as log events occur on the web app. Any information written to files ending in .txt, .log, or .htm that are stored in the /LogFiles directory (d:/home/logfiles) is streamed to the local console.

To filter specific events, such as errors, use the **--Filter** parameter. For example:

```
az webapp log tail --name webappname --resource-group myResourceGroup --filter Error
```

To filter specific log types, such as HTTP, use the **--Path** parameter. For example:

```
az webapp log tail --name webappname --resource-group myResourceGroup --path http
```

NOTE

If you have not installed the Azure Command-Line Interface, or have not configured it to use your Azure Subscription, see [How to Use Azure Command-Line Interface](#).

How to: Understand diagnostics logs

Application diagnostics logs

Application diagnostics stores information in a specific format for .NET applications, depending on whether you store logs to the file system, table storage, or blob storage. The base set of data stored is the same across all three storage types - the date and time the event occurred, the process ID that produced the event, the event type (information, warning, error), and the event message.

File system

Each line logged to the file system or received using streaming is in the following format:

```
{Date} {PID}[{process ID}] {event type/level} {message}
```

For example, an error event would appear similar to the following sample:

```
2014-01-30T16:36:59 PID[3096] Error      Fatal error on the page!
```

Logging to the file system provides the most basic information of the three available methods, providing only the time, process ID, event level, and message.

Table storage

When logging to table storage, additional properties are used to facilitate searching the data stored in the table as well as more granular information on the event. The following properties (columns) are used for each entity (row) stored in the table.

PROPERTY NAME	VALUE/FORMAT
PartitionKey	Date/time of the event in yyyyMMddHH format
RowKey	A GUID value that uniquely identifies this entity
Timestamp	The date and time that the event occurred
EventTickCount	The date and time that the event occurred, in Tick format (greater precision)

PROPERTY NAME	VALUE/FORMAT
ApplicationName	The web app name
Level	Event level (for example, error, warning, information)
EventId	<p>The event ID of this event</p> <p>Defaults to 0 if none specified</p>
InstanceId	Instance of the web app that the even occurred on
Pid	Process ID
Tid	The thread ID of the thread that produced the event
Message	Event detail message

Blob storage

When logging to blob storage, data is stored in comma-separated values (CSV) format. Similar to table storage, additional fields are logged to provide more granular information about the event. The following properties are used for each row in the CSV:

PROPERTY NAME	VALUE/FORMAT
Date	The date and time that the event occurred
Level	Event level (for example, error, warning, information)
ApplicationName	The web app name
InstanceId	Instance of the web app that the event occurred on
EventTickCount	The date and time that the event occurred, in Tick format (greater precision)
EventId	<p>The event ID of this event</p> <p>Defaults to 0 if none specified</p>
Pid	Process ID
Tid	The thread ID of the thread that produced the event
Message	Event detail message

The data stored in a blob would look similar to the following example:

```
date,level,applicationName,instanceId,eventTickCount,eventId,pid,tid,message
2014-01-30T16:36:52>Error,mywebapp,6ee38a,635266966128818593,0,3096,9,An error occurred
```

NOTE

The first line of the log contains the column headers as represented in this example.

Failed request traces

Failed request traces are stored in XML files named **fr#####.xml**. To make it easier to view the logged information, an XSL stylesheet named **freb.xsl** is provided in the same directory as the XML files. If you open one of the XML files in Internet Explorer, Internet Explorer uses the XSL stylesheet to provide a formatted display of the trace information, similar to the following example:

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

Detailed error logs

Detailed error logs are HTML documents that provide more detailed information on HTTP errors that have

occurred. Since they are simply HTML documents, they can be viewed using a web browser.

Web server logs

The web server logs are formatted using the [W3C extended log file format](#). This information can be read using a text editor or parsed using utilities such as [Log Parser](#).

NOTE

The logs produced by Azure web apps do not support the **s-computername**, **s-ip**, or **cs-version** fields.

Next steps

- [How to Monitor Web Apps](#)
- [Troubleshooting Azure web apps in Visual Studio](#)
- [Analyze web app Logs in HDInsight](#)

NOTE

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service](#), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

Manage an App Service plan in Azure

1/2/2018 • 3 min to read • [Edit Online](#)

An [App Service plan](#) provides the resources an App Service app needs to run. This how-to guide shows how to manage an App Service plan.

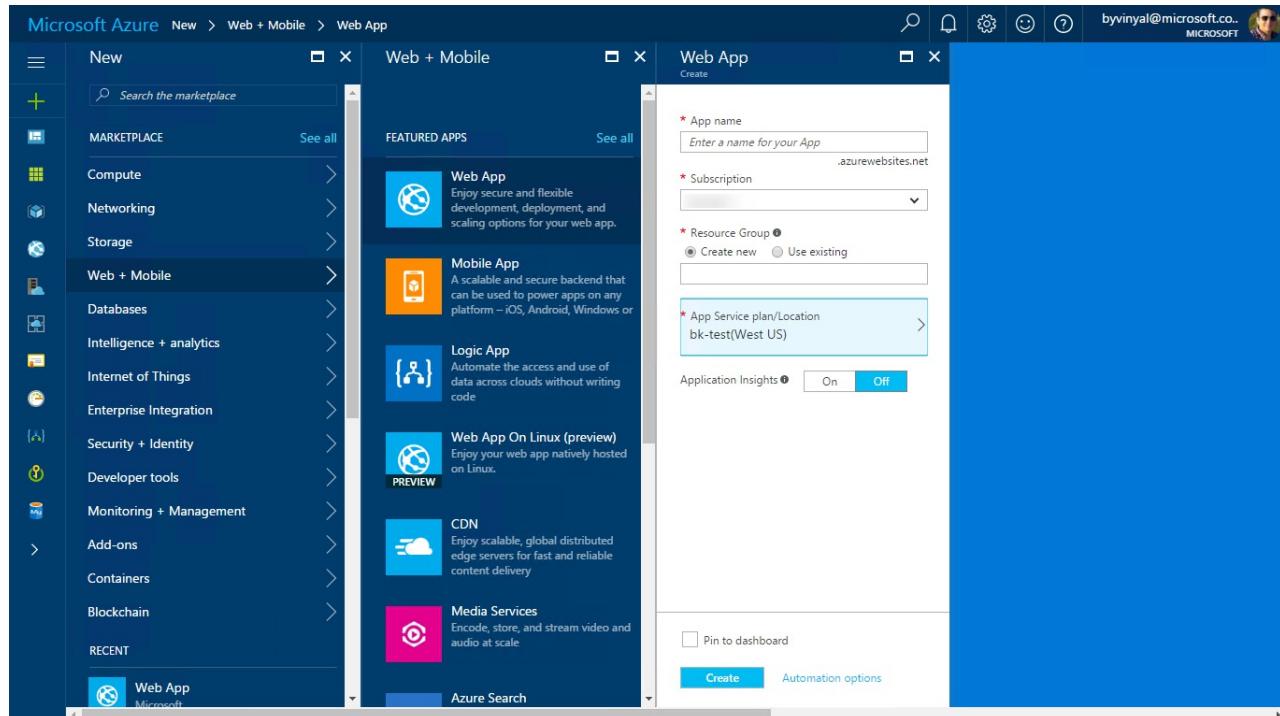
Create an App Service plan

TIP

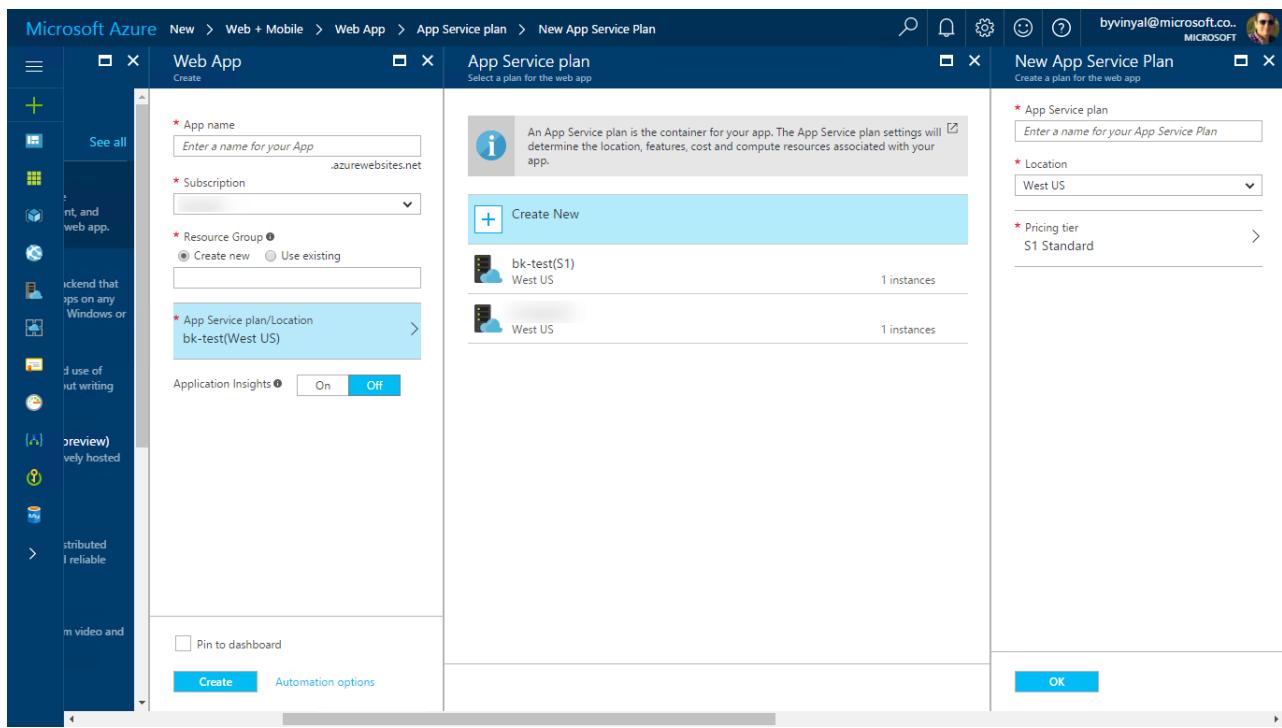
If you have an App Service Environment, see [Create an App Service plan in an App Service Environment](#).

You can create an empty App Service plan or as part of app creation.

In the [Azure portal](#), click **New > Web + mobile**, and then select **Web App** or other App Service app kind.



You can then select an existing App Service plan or create a plan for the new app.



To create an App Service plan, click **[+] Create New**, type the **App Service plan** name, and then select an appropriate **Location**. Click **Pricing tier**, and then select an appropriate pricing tier for the service. Select **View all** to view more pricing options, such as **Free** and **Shared**.

After you have selected the pricing tier, click the **Select** button.

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*.

To move an app to another plan, navigate to the app that you want to move in the [Azure portal](#).

In the **Menu**, look for the **App Service Plan** section.

Select **Change App Service plan** to start the process.

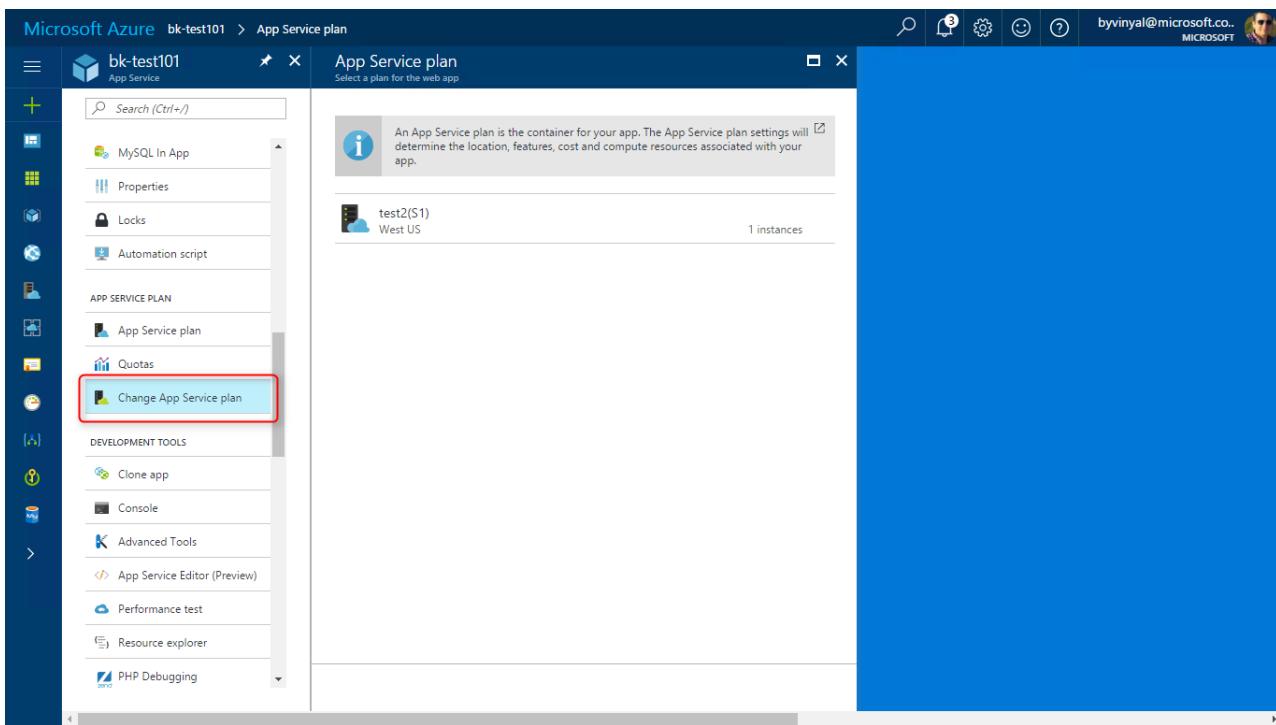
Change App Service plan opens the **App Service plan** selector. Select an existing plan to move this app into.

IMPORTANT

The **Select App Service plan** page is filtered by the following criteria:

- Exists in the same resource group
- Exists in the same geographical region
- Exists in the same webspace

A *webspace* is a logical construct in App Service that defines a grouping of server resources. A geographical region (such as West US) contains many webspaces in order to allocate customers using App Service. Currently, App Service resources aren't able to be moved between webspaces.



NOTE

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

Each plan has its own pricing tier. For example, moving a site from a **Free** tier to a **Standard** tier, enables all apps assigned to it to use the features and resources of the **Standard** tier. However, moving an app from a higher tiered plan to a lower tiered plan means that you no longer have access to certain features. If your app uses a feature that is not available in the target plan, you get an error that shows which feature is in use that is not available. For example, if one of your apps uses SSL certificates, you might see the 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'.
```

In this case, you need to scale up the pricing tier of the target plan to **Basic** or higher, or you need to remove all SSL connections to your app, before you can move the app to the target plan.

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 that you can read about at [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 choose to keep the plan instead, you should change the plan to **Free** tier so that you don't get charged.

IMPORTANT

App Service plans that have no apps associated to them still incur charges since they continue to reserve the configured VM instances.

Next steps

[Scale up an app in Azure](#)

Back up your app in Azure

11/20/2017 • 5 min 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 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 \(Preview\)](#)
- [Azure Database for PostgreSQL \(Preview\)](#)
- [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 the [links](#) at the end of this article.
- Backups can be up to 10 GB of app and database content. If the backup size exceeds this limit, you get an error.

Create a manual backup

1. In the [Azure portal](#), navigate to your app's page, select **Backups**. The **Backups** page is displayed.

The screenshot shows the Azure portal interface for an App Service named 'contoso-backup'. The top navigation bar includes a search bar labeled 'Search (Ctrl+ /)'. On the left, a sidebar titled 'SETTINGS' lists 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](#).



2. In the **Backup** page, Click **Configure**

The screenshot shows the 'Backup' configuration page. At the top, there's a blue cloud icon and the word 'Backup'. Below that, a message says 'Configure backup to create restorable archive copies of your apps content, configuration and database.' with a 'Learn more' link. A note message states 'Backup Not Configured. Configure you backup by setting up a storage account, schedule and select databases to be backed up for safe keeping and disaster recovery.' At the bottom, a large blue button with a gear icon and the word 'Configure' is highlighted with a red box.

3. In the **Backup Configuration** page, click **Storage: Not configured** to configure a storage account.

Backup Configuration

X



Backup Storage

Select the target container to store your app backup.

Storage Settings
Storage not configured



Backup Schedule

Configure the schedule for your app backup.

Scheduled backup



Backup Database

Select the databases you to include with your backup. The backup database list is based on the apps configured connection strings.



INCLUDE IN BACKUP

CONNECTION STRING NAME

DATABASE TYPE

No supported connection strings of type SQL Database or MySQL found configured in app.

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**.

NAME	TYPE	RESOURCE GROUP	LOCATION
ahmedbackupstorage	Standard-LRS	snapshot-rg	West US
...
...
...
...

NAME	LAST MODIFIED
backups	Thu May 04 2017 13:44:21 GMT-0700 (Pacific Daylight Time)

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 Configuration



Backup Storage

Select the target container to store your app backup.

Storage Settings
backups



Backup Schedule

Configure the schedule for your app backup.

Scheduled backup



Backup Database

Select the databases you to include with your backup. The backup database list is based on the apps configured connection strings.

<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.		

Save

Discard

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.

6. In the **Backup Configuration** page, click **Save**.

7. In the **Backups** page, click **Backup**.

Refresh Reset Configuration



Backup

Configure backup to create restorable archive copies of your apps content, configuration and database. [Learn more](#)



Backup configured, Schedule for backup is not configured, Configure scheduled backup to automatically take backups.



Configure



Backup



Restore

STATUS

BACKUP TIME



Created

Thursday, June 18, 2015, 11:00AM UTC

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**.

The screenshot shows the 'Backup Configuration' page. It has two main sections: 'Backup Storage' and 'Backup Schedule'.
In the 'Backup Storage' section, there is a note: 'Select the target container to store your app backup.' Below it are 'Storage Settings' and 'backups' links.
In the 'Backup Schedule' section, there is a note: 'Configure the schedule for your app backup.' Below it is a switch labeled 'Scheduled backup' with options 'On' (highlighted with a red box) and 'Off'.
The entire screenshot is framed by a thick black border.

2. Backup schedule options will show up, set **Scheduled Backup** to **On**, then configure the backup schedule as desired and click **OK**.

The screenshot shows the 'Backup Schedule' configuration page. It includes fields for scheduling:
- 'Scheduled backup' switch (set to 'On', highlighted with a red box).
- 'Backup Every' field (set to '1') with dropdowns for 'Days' and 'Hours'.
- 'Start backup schedule from' date and time fields (set to '2017-06-08 5:13:55 PM UTC -07:00').
- 'Retention (Days)' field (set to '30').
- 'Keep at least one backup' switch (set to 'Yes').
The entire screenshot is framed by a thick black border.

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).

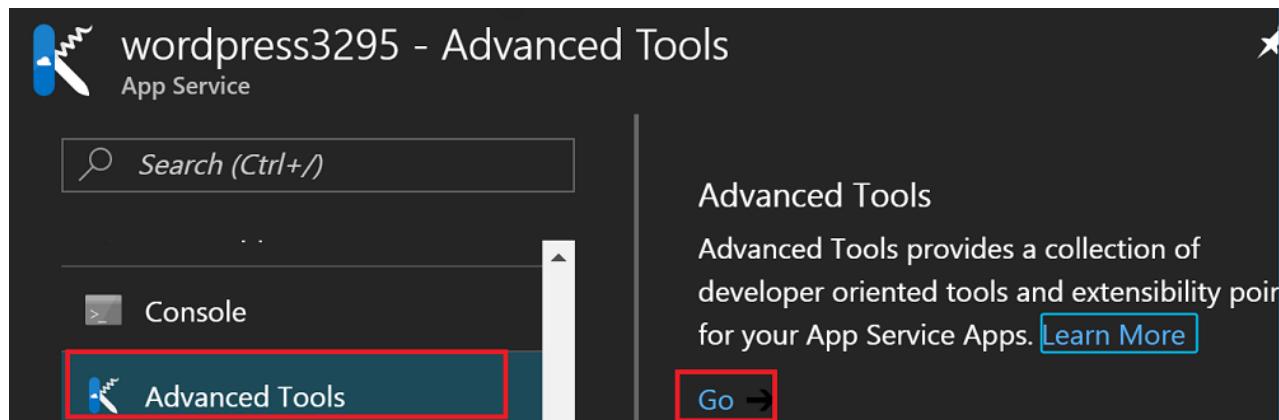
- You don't want to back up the log files.

Partial backups allow you choose exactly which files you want to back up.

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.

An easy way to access your files is to use Kudu. Click **Advanced Tools -> Go** setting for your web app to access Kudu.



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

11/20/2017 • 2 min 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 \(Preview\)](#)
- [Azure Database for PostgreSQL \(Preview\)](#)
- [MySQL in-app](#)

Restoring from backups is available to apps running in **Standard** and **Premium** tier. For information about scaling up your app, see [Scale up an app in Azure](#). **Premium** tier allows a greater number of daily backups to be performed than **Standard** tier.

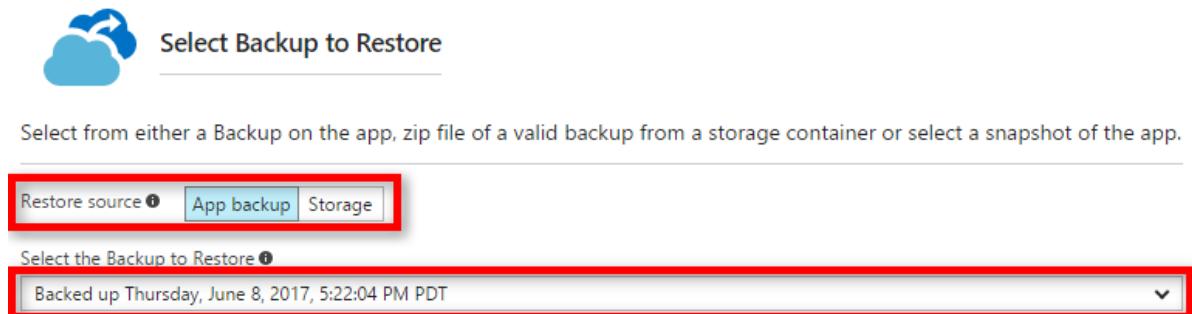
Restore an app from an existing backup

1. On the **Settings** page of your app in the Azure portal, click **Backups** to display the **Backups** page. Then click **Restore**.

The screenshot shows the 'Backups' page for an app named 'contoso-backup'. The left sidebar has a red box around the 'Backups' link under the 'SETTINGS' section. The main area has a 'Backup' section with a warning icon and text about configuration. Below it are 'Configure' and 'Restore' buttons, with 'Restore' being redboxed. A table at the bottom lists a single backup entry with a checkmark, timestamp, and size.

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.

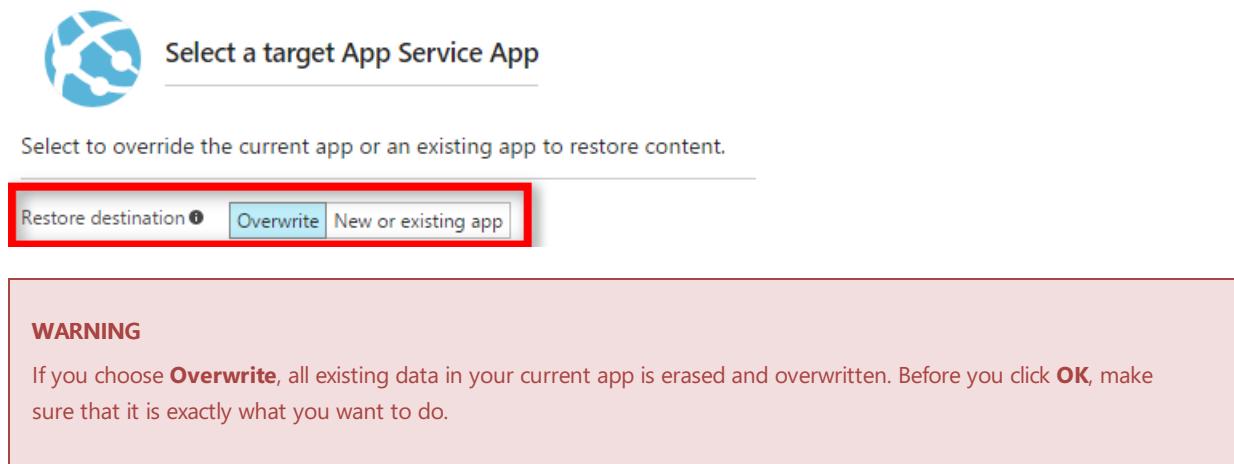
Restore source ⓘ App backup Storage

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.

You can select **Existing App** to restore the app backup to another app in the same resource group. Before you use this option, you should have already created another app in your resource group with mirroring database configuration to the one defined in the app backup. You can also Create a **New** app to restore your content to.

4. Click **OK**.

Download or delete a backup from a storage account

1. From the main **Browse** page of the Azure portal, select **Storage accounts**. A list of your existing storage accounts is displayed.
2. Select the storage account that contains the backup that you want to download or delete. The page for the storage account is displayed.
3. In the storage account page, select the container you want

The screenshot shows the Azure Storage Account - Blob blade for the storage account 'cephalinstorage4'. At the top, there are navigation icons for Settings, Delete, Container, and Refresh. Below that is the 'Essentials' section with a collapsible arrow. The essentials information includes:

- Resource group: [cephalin-appwithsql](#)
- Status: Primary: Available
- Location: West Europe
- Subscription name: [Visual Studio Ultimate with MSDN](#)
- Subscription ID: [REDACTED]

On the right side of the essentials section are four small icons: a key, a lightning bolt, two people, and a tag.

Below the essentials section is a 'Performance/Access tier' section with a pencil icon, showing Standard/Hot. It also lists Replication (Locally-redundant storage (LRS)) and the Blob service endpoint (<https://cephalinstorage4.blob.core.windows..>).

At the bottom of the essentials section is a link to 'All settings →'.

Below the essentials section is a search bar with the placeholder 'Search containers by prefix'.

The main content area displays a table of containers:

NAME	URL	LAST MODIFIED	...
backups	https://cephalinstorage4.blob.core.windows.net/backups	7/6/2016, 2:00:16 PM	...

4. Select backup file you want to download or delete.

The screenshot shows the Azure Storage Explorer interface. At the top, there's a header bar with icons for Refresh, Delete container, Properties, and Access policy. Below the header is a search bar with the placeholder text "Search blobs by prefix (case-sensitive)". The main area displays a table of blobs in the "backups" container. The columns are NAME, MODIFIED, BLOB TYPE, and SIZE. There are three blobs listed:

NAME	MODIFIED	BLOB TYPE	SIZE
cephalin-appwithsql_201607061211.log	7/6/2016, 2:15:58..	Block blob	272 B
cephalin-appwithsql_201607061211.xml	7/6/2016, 2:15:58..	Block blob	793 B
cephalin-appwithsql_201607061211.zip	7/6/2016, 2:15:58..	Block blob	151.11 KB

5. Click **Download** or **Delete** depending on what you want to do.

Monitor a restore operation

To see details about the success or failure of the app restore operation, navigate to the **Activity Log** page in the Azure portal.

Scroll down to find the desired restore operation and click to select it.

The details page displays the available information related to the restore operation.

Automate with scripts

You can automate backup management with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

For samples, see:

- [Azure CLI samples](#)
- [Azure PowerShell samples](#)

Azure App Service App Cloning Using PowerShell

9/19/2017 • 5 min to read • [Edit Online](#)

With the release of Microsoft Azure PowerShell version 1.1.0 a new option has been added to New-AzureRMWebApp that would give the user the ability to clone an existing Web App to a newly created app in a different region or in the same region. This will enable customers to deploy a number of apps across different regions quickly and easily.

App cloning is currently only supported for premium tier app service plans. The new feature uses the same limitations as Web Apps Backup feature, see [Back up a web app in Azure App Service](#).

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

Cloning an existing App

Scenario: An existing web app in South Central US region, the user would like to clone the contents to a new web app in North Central US region. This can be accomplished by using the Azure Resource Manager version of the PowerShell cmdlet to create a new web app with the -SourceWebApp option.

Knowing the resource group name that contains the source web app, we can use the following PowerShell command to get the source web app's information (in this case named source-webapp):

```
$srcapp = Get-AzureRmWebApp -ResourceGroupName SourceAzureResourceGroup -Name source-webapp
```

To create a new App Service Plan, we can use New-AzureRmAppServicePlan command as in the following example

```
New-AzureRmAppServicePlan -Location "South Central US" -ResourceGroupName DestinationAzureResourceGroup -Name NewAppServicePlan -Tier Premium
```

Using the New-AzureRmWebApp command, we can create the new web app in the North Central US region, and tie it to an existing premium tier App Service Plan, moreover we can use the same resource group as the source web app, or define a new resource group, the following demonstrates that:

```
$destapp = New-AzureRmWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp
```

To clone an existing web app including all associated deployment slots, the user will need to use the IncludeSourceWebAppSlots parameter, the following PowerShell command demonstrates the use of that parameter with the New-AzureRmWebApp command:

```
$destapp = New-AzureRmWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -IncludeSourceWebAppSlots
```

To clone an existing web app within the same region, the user will need to create a new resource group and a new app service plan in the same region, and then using the following PowerShell command to clone the web app

```
$destapp = New-AzureRmWebApp -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 web app in South Central US region, the user would like to clone the contents to a new web app to an existing App Service Environment (ASE).

Knowing the resource group name that contains the source web app, we can use the following PowerShell command to get the source web app's information (in this case named source-webapp):

```
$srcapp = Get-AzureRmWebApp -ResourceGroupName SourceAzureResourceGroup -Name source-webapp
```

Knowing the ASE's name, and the resource group name that the ASE belongs to, the user can use the New-AzureRmWebApp command to create the new web app in the existing ASE, the following demonstrates that:

```
$destapp = New-AzureRmWebApp -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 in the case of creating an app in an ASE it will be ignored.

Cloning an existing App Slot

Scenario: The user would like to clone an existing Web App Slot to either a new Web App or a new Web App slot. The new Web App can be in the same region as the original Web App slot or in a different region.

Knowing the resource group name that contains the source web app, we can use the following PowerShell command to get the source web app slot's information (in this case named source-webappslot) tied to Web App source-webapp:

```
$srcappslot = Get-AzureRmWebAppSlot -ResourceGroupName SourceAzureResourceGroup -Name source-webapp -Slot source-webappslot
```

The following demonstrates creating a clone of the source web app to a new web app:

```
$destapp = New-AzureRmWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcappslot
```

Configuring Traffic Manager while cloning a App

Creating multi-region web apps and configuring Azure Traffic Manager to route traffic to all these web apps, is an important scenario to insure that customers' apps are highly available, when cloning an existing web app you have the option to connect both web apps to either a new traffic manager profile or an existing one - note that only Azure Resource Manager version of Traffic Manager is supported.

Creating a new Traffic Manager profile while cloning a App

Scenario: The user would like to clone an web app to another region, while configuring an Azure Resource Manager traffic manager profile that include both web apps. The following demonstrates creating a clone of the source web app to a new web app while configuring a new Traffic Manager profile:

```
$destapp = New-AzureRmWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "South Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -TrafficManagerProfileName newTrafficManagerProfile
```

Adding new cloned Web App to an existing Traffic Manager profile

Scenario: The user already have an Azure Resource Manager traffic manager profile that he would like to add both web apps as endpoints. To do so, we first need to assemble the existing traffic manager profile id, we will need the subscription id, resource group name and the existing traffic manager profile name.

```
$TMProfileID = "/subscriptions/<Your subscription ID goes here>/resourceGroups/<Your resource group name goes here>/providers/Microsoft.TrafficManagerProfiles/ExistingTrafficManagerProfileName"
```

After having the traffic manger id, the following demonstrates creating a clone of the source web app to a new web app while adding them to an existing Traffic Manager profile:

```
$destapp = New-AzureRmWebApp -ResourceGroupName <Resource group name> -Name dest-webapp -Location "South Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -TrafficManagerProfileId $TMProfileID
```

Current Restrictions

This feature is currently in preview, we are working to add new capabilities over time, the following list are the known restrictions on the current version 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 web app
- Easy Auth settings are not cloned
- Kudu Extension are not cloned
- TiP rules are not cloned
- Database content are not cloned

References

- [Web App Cloning](#)
- [Back up a web app in Azure App Service](#)
- [Azure Resource Manager support for Azure Traffic Manager Preview](#)
- [Introduction to App Service Environment](#)
- [Using Azure PowerShell with Azure Resource Manager](#)

Move an Azure Web App to another resource group

10/19/2017 • 1 min to read • [Edit Online](#)

You can move a Web App and/or its related resources to another resource group in the same subscription, or to a resource group in a different subscription. This is done as part of standard resource management in Azure. For more information, see [Move Azure resources to new subscription or resource group](#).

Limitations when moving within the same subscription

When moving a Web App *within the same subscription*, you cannot move the uploaded SSL certificates. However, you can move a Web App to the new resource group without moving its uploaded SSL certificate, and your app's SSL functionality still works.

If you want to move the SSL certificate with the Web App, follow these steps:

1. Delete the uploaded certificate from the Web App.
2. Move the Web App.
3. Upload the certificate to the moved Web App.

Limitations when moving across subscriptions

When moving a Web App *across subscriptions*, the following limitations apply:

- The destination resource group must not have any existing App Service resources. App Service resources include:
 - Web Apps
 - App Service plans
 - Uploaded or imported SSL certificates
 - App Service Environments
- All App Service resources in the resource group must be moved together.
- App Service resources can only be moved from the resource group in which they were originally created. If an App Service resource is no longer in its original resource group, it must be moved back to that original resource group first, and then it can be moved across subscriptions.

Run Background tasks with WebJobs in Azure App Service

11/22/2017 • 5 min to read • [Edit Online](#)

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.

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](#).

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 scm (deployment) site or to the web app's pages in the portal reset the timer. Requests to the actual site don't reset the timer. If your app runs continuous or scheduled 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

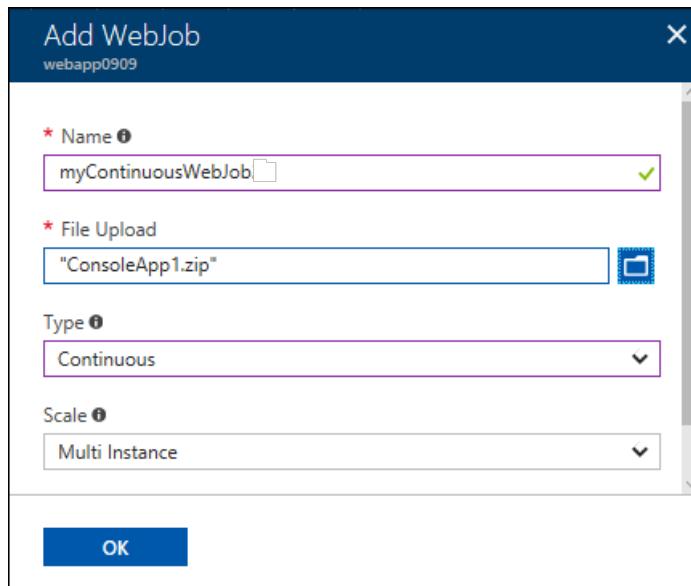
1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. Select **WebJobs**.

The screenshot shows the Azure portal interface for an App Service named 'WebApp0909'. The left sidebar contains several settings: Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' option is highlighted with a red box.

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 red-highlighted 'Add' button, Refresh, Logs, Delete, and Properties options. The left sidebar lists WebJobs, Push, MySQL In App, Properties, Locks, and Automation script. The main area is titled 'WebJobs' and contains the text: 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' Below this, a table header is shown with columns: 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 - WebJobs". On the left, a sidebar titled "SETTINGS" 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 blue selection bar. 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 above, but with a context menu open over the "myContinuousW..." row. The menu items are: Logs (with a small icon), Delete (with a trash icon), Stop (with a square icon), and Properties (with a gear icon). The "Stop" item is highlighted with a red rectangular box.

Create a manually triggered WebJob

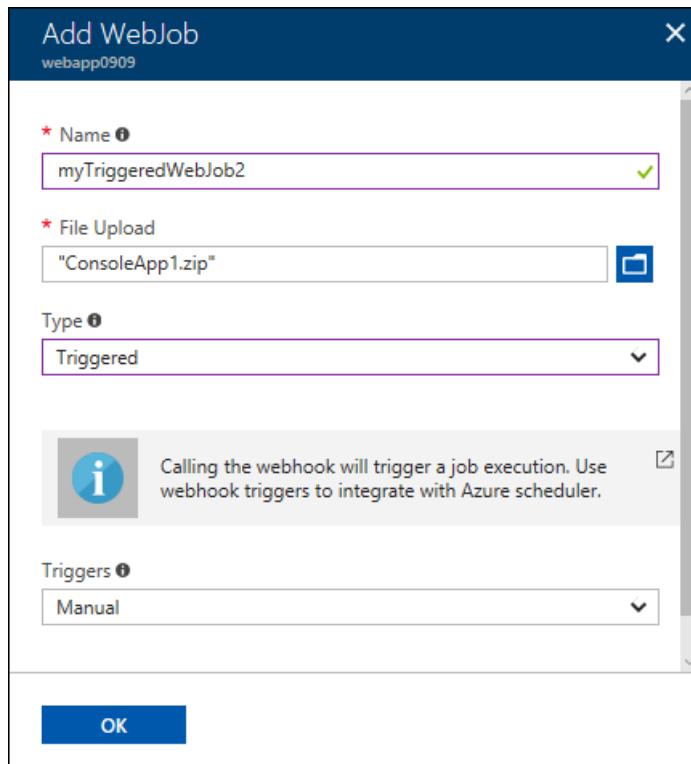
1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. Select **WebJobs**.

The screenshot shows the Azure portal interface for a web application named 'WebApp0909 - WebJobs'. The left sidebar contains a search bar and a list of settings: Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' item is highlighted with a red box.

3. In the **WebJobs** page, select **Add**.

The screenshot shows the 'WebJobs' page for the same application. The left sidebar lists WebJobs, Push, MySQL In App, Properties, Locks, and Automation script. The main area has a heading 'WebJobs' with a sub-section about running scripts or programs. A table header for 'NAME', 'TYPE', 'STATUS', and 'SCHEDULE' is shown, followed by the message 'You haven't added any WebJobs. Click ADD to get started.' The 'Add' button in the top navigation bar is highlighted with a red box.

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	n/a

Right-click context menu for myTriggeredWeb...:

- Logs
- Delete
- Run** (highlighted with a red box)
- Properties

Create a scheduled WebJob

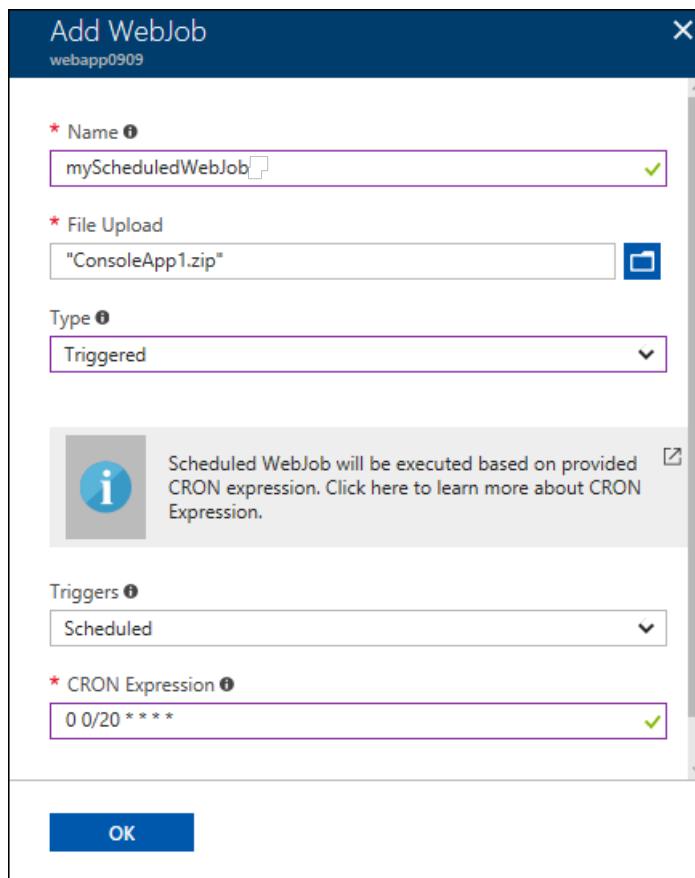
- In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
- Select **WebJobs**.

The screenshot shows the Azure portal interface for a web application named 'WebApp0909 - WebJobs'. The left sidebar contains a search bar and a list of settings: Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' item is highlighted with a red box.

3. In the **WebJobs** page, select **Add**.

The screenshot shows the 'WebJobs' page for the same application. The left sidebar lists WebJobs, Push, MySQL In App, Properties, Locks, and Automation script. The main area has a heading 'WebJobs' with a sub-section about running scripts or programs. A table header for 'NAME', 'TYPE', 'STATUS', and 'SCHEDULE' is shown, followed by the message 'You haven't added any WebJobs. Click ADD to get started.' The 'Add' button in the top navigation bar 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.

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Ready	0 0/15 * * *
myTriggeredWeb...	Triggered	Ready	n/a
myContinuousW...	Continuous	Pending Restart	n/a

CRON expressions

A [CRON expression](#) is composed of six fields: `{second} {minute} {hour} {day} {month} {day of the week}`. Here are some examples:

- Every 15 minutes: `0 */15 * * * *`
- Every hour (that is, whenever the count of minutes is 0): `0 0 * * * *`
- Every hour from 9 AM to 5 PM: `0 0 9-17 * * *`
- At 9:30 AM every day: `0 30 9 * * *`
- At 9:30 AM every weekday: `0 30 9 * * 1-5`

You can enter the CRON 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 * * * *"
}
```

NOTE

When you deploy a WebJob from Visual Studio, mark your `settings.job` file properties as **Copy if newer**.

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

11/22/2017 • 6 min to read • [Edit Online](#)

Overview

This topic 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](#).

When Visual Studio deploys a WebJobs-enabled Console Application project, it performs two tasks:

- Copies runtime files to the appropriate folder in the web app (*App_Data/jobs/continuous* for continuous WebJobs, *App_Data/jobs/triggered* for scheduled and on-demand WebJobs).
- Sets up [Azure Scheduler jobs](#) for WebJobs that are scheduled to run at particular times. (This is not needed for continuous 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.](#)

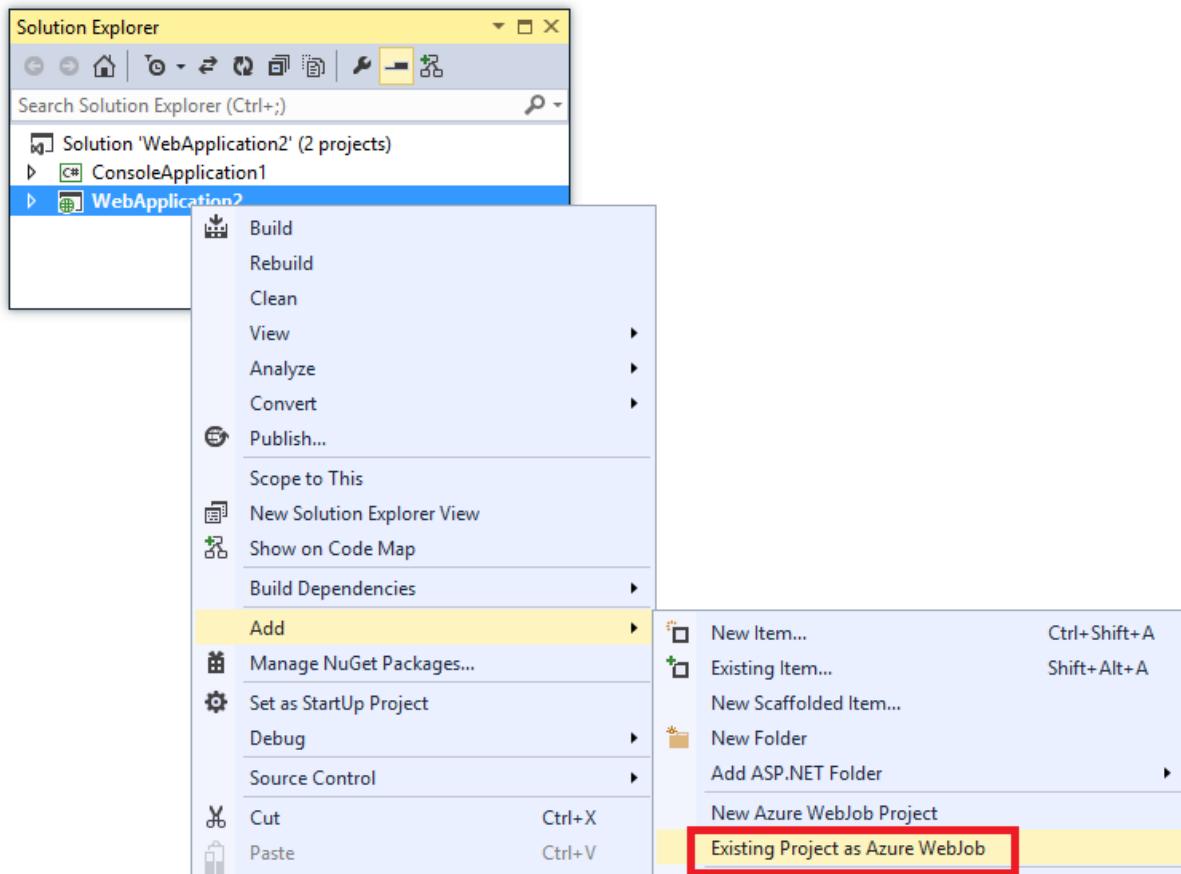
Configure an existing Console Application project so that it automatically deploys as a WebJob when you deploy a web project. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

- [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.

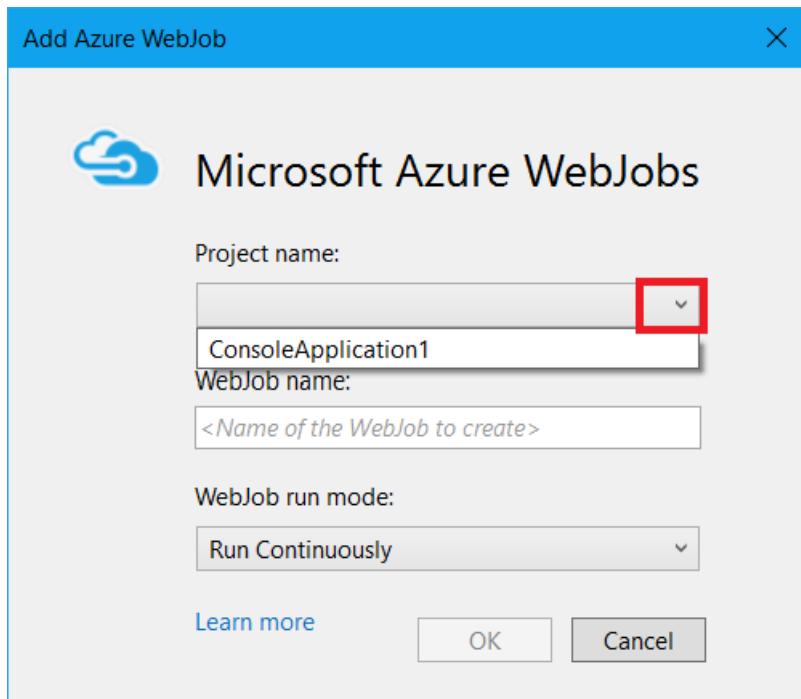
Enable automatic WebJobs deployment with a web project

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.

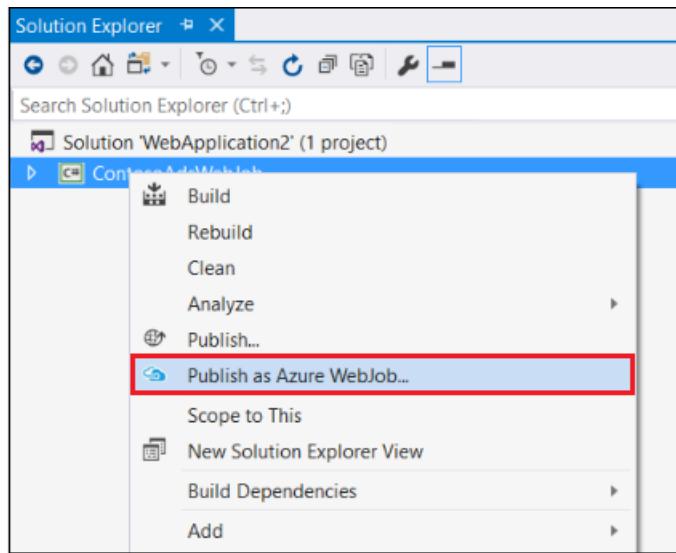
2. In the **Project name** drop-down list, select the Console Application project to add as a WebJob.



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**

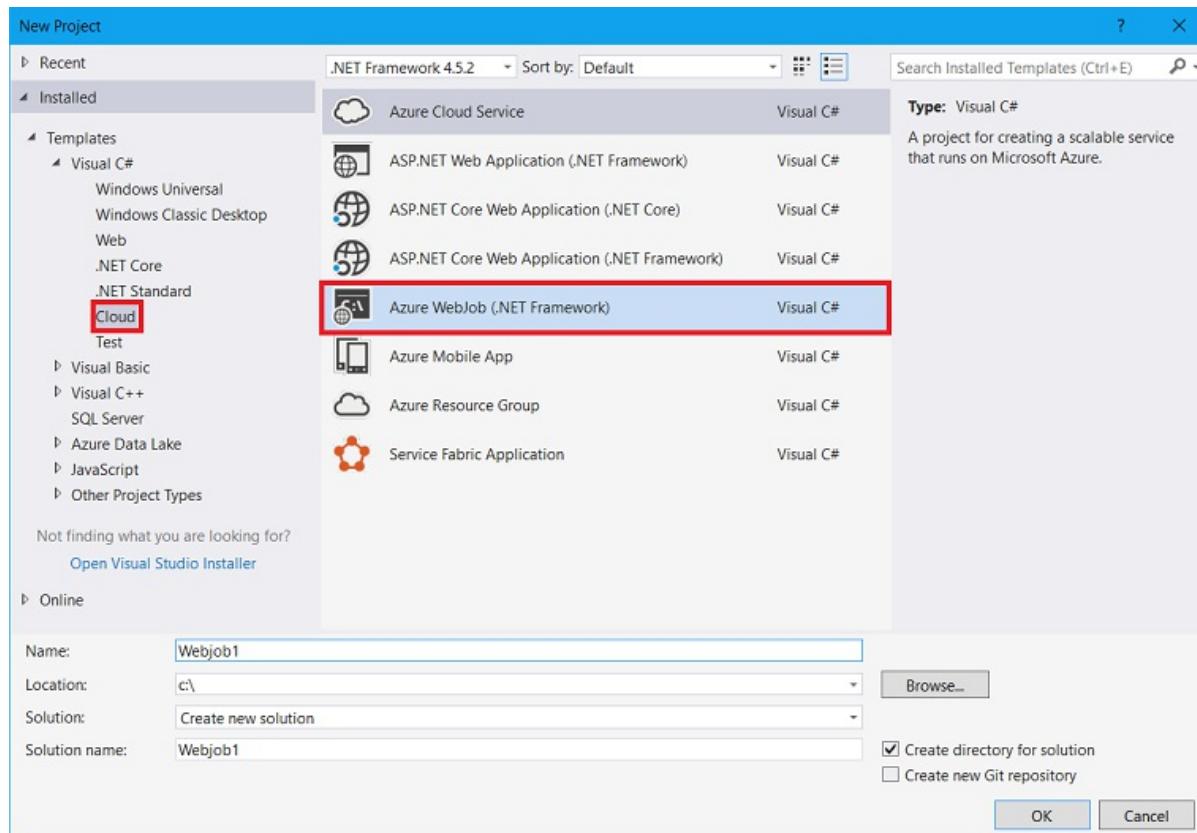
Create a project that is configured to deploy automatically as a WebJob when a web project in the same solution is deployed. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

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

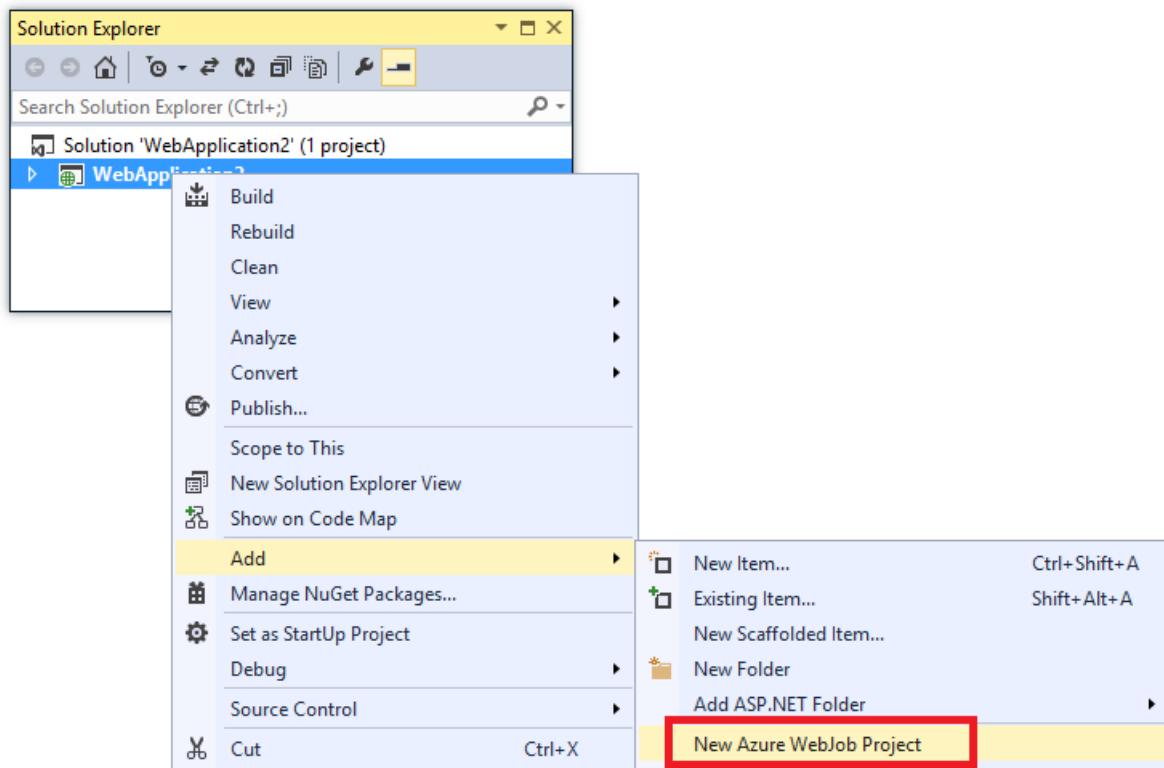
1. Click **File > New Project**, and then in the **New Project** dialog box click **Cloud > Azure WebJob (.NET Framework)**.



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**.

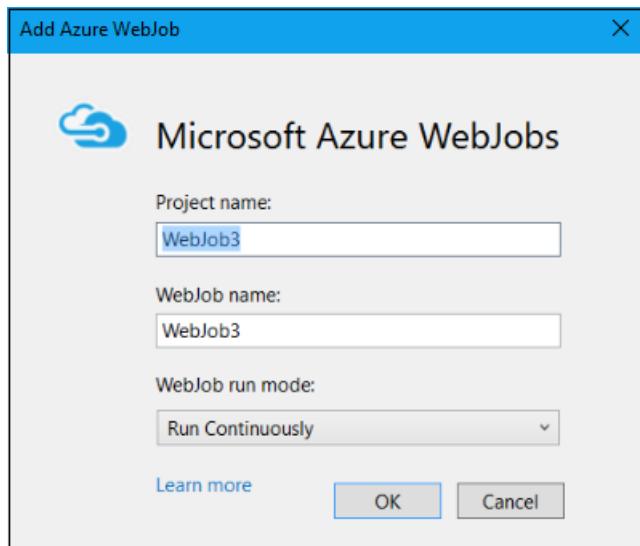


The [Add Azure WebJob](#) dialog box appears.

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 <http://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"  
    }  
  ]  
}
```

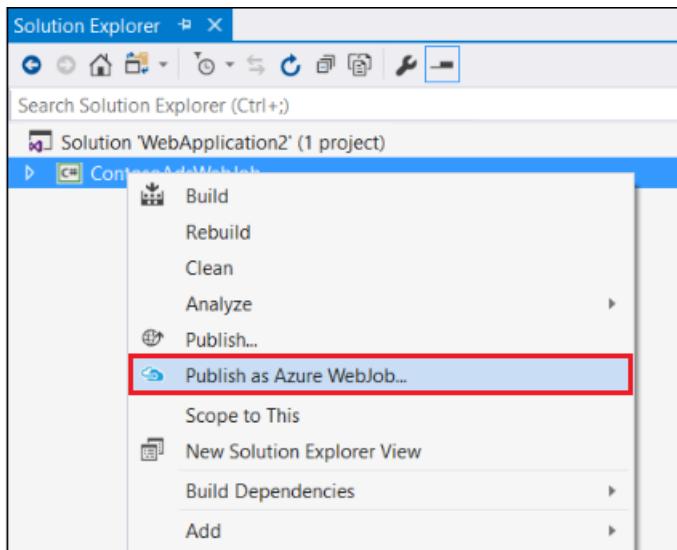
You can edit this file directly, and Visual Studio provides IntelliSense. The file schema is stored at <http://schemastore.org> and can be viewed there.

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.

Azure subscription and service limits, quotas, and constraints

12/12/2017 • 58 min to read • [Edit Online](#)

This document lists some of the most common Microsoft Azure limits, which are also sometimes called quotas. This document doesn't currently cover all Azure services. Over time, the list will be expanded and updated to cover more of the platform.

Please visit [Azure Pricing Overview](#) to learn more about Azure pricing. There, you can estimate your costs using the [Pricing Calculator](#) or by visiting the pricing details page for a service (for example, [Windows VMs](#)). For tips to help manage your costs, see [Prevent unexpected costs with Azure billing and cost management](#).

NOTE

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 is no **Maximum Limit** column, then the resource doesn't have adjustable limits.

[Free Trial subscriptions](#) are not 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 Azure Free Trial to Pay-As-You-Go](#) and [Free Trial subscription FAQ](#).

Limits and the Azure Resource Manager

It is now possible to combine multiple Azure resources in to a single Azure Resource Group. When using Resource Groups, limits that once were global become managed at a regional level with the Azure Resource Manager. For more information about Azure Resource Groups, see [Azure Resource Manager overview](#).

In the limits below, a new table has been added to reflect any differences in limits when using the Azure Resource Manager. For example, there is a **Subscription Limits** table and a **Subscription Limits - Azure Resource Manager** table. When a limit applies to both scenarios, it is only shown in the first table. Unless otherwise indicated, limits are global across all regions.

NOTE

It is important to emphasize that quotas for resources in Azure Resource Groups are per-region accessible by your subscription, and are not per-subscription, as the service management quotas are. Let's use vCPU quotas as an example. If you need to request a quota increase with support for vCPUs, you need to decide how many vCPUs you want to use in which regions, and then make a specific request for Azure Resource Group vCPU quotas for the amounts and regions that you want. Therefore, if you need to use 30 vCPUs in West Europe to run your application there, you should specifically request 30 vCPUs in West Europe. But you will not have a vCPU quota increase in any other region -- only West Europe will have the 30-vCPU quota.

As a result, you may find it useful to consider deciding what your Azure Resource Group quotas need to be for your workload in any one region, and request that amount in each region into which you are considering deployment. See [troubleshooting deployment issues](#) for more help discovering your current quotas for specific regions.

Service-specific limits

- [Active Directory](#)

- [API Management](#)
- [App Service](#)
- [Application Gateway](#)
- [Application Insights](#)
- [Automation](#)
- [Azure Cosmos DB](#)
- [Azure Event Grid](#)
- [Azure Redis Cache](#)
- [Backup](#)
- [Batch](#)
- [BizTalk Services](#)
- [CDN](#)
- [Cloud Services](#)
- [Container Instances](#)
- [Container Registry](#)
- [Data Factory](#)
- [Data Lake Analytics](#)
- [Data Lake Store](#)
- [Database Migration Service](#)
- [DNS](#)
- [Event Hubs](#)
- [IoT Hub](#)
- [IoT Hub Device Provisioning Service](#)
- [Key Vault](#)
- [Log Analytics / Operational Insights](#)
- [Media Services](#)
- [Mobile Engagement](#)
- [Mobile Services](#)
- [Monitor](#)
- [Multi-Factor Authentication](#)
- [Networking](#)
- [Network Watcher](#)
- [Notification Hub Service](#)
- [Resource Group](#)
- [Scheduler](#)
- [Search](#)
- [Service Bus](#)
- [Site Recovery](#)
- [SQL Database](#)
- [SQL Data Warehouse](#)
- [Storage](#)
- [StorSimple System](#)
- [Stream Analytics](#)
- [Subscription](#)
- [Traffic Manager](#)
- [Virtual Machines](#)

- [Virtual Machine Scale Sets](#)

Subscription limits

Subscription limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Cores per subscription ¹	20	10,000
Co-administrators per subscription	200	200
Storage accounts per subscription ²	200	250
Cloud services per subscription	20	200
Local networks per subscription	10	500
SQL Database servers per subscription	6	150
DNS servers per subscription	9	100
Reserved IPs per subscription	20	100
Hosted service certificates per subscription	400	400
Affinity groups per subscription	256	256

¹Extra Small instances count as one core towards the core limit despite using a partial core.

²This includes both Standard and Premium storage accounts. If you require more than 200 storage accounts, make a request through [Azure Support](#). The Azure Storage team will review your business case and may approve up to 250 storage accounts.

Subscription limits - Azure Resource Manager

The following limits apply when using the Azure Resource Manager and Azure Resource Groups. Limits that have not changed with the Azure Resource Manager are not listed below. Please refer to the previous table for those limits.

For information about handling limits on Resource Manager requests, see [Throttling Resource Manager requests](#).

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
VMs per subscription	10,000 ¹ per Region	10,000 per Region
VM total cores per subscription	20 ¹ per Region	Contact support
VM per series (Dv2, F, etc.) cores per subscription	20 ¹ per Region	Contact support
Co-administrators per subscription	Unlimited	Unlimited
Storage accounts per subscription	200	200 ²
Resource Groups per subscription	800	800

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Availability Sets per subscription	2,000 per Region	2,000 per Region
Resource Manager API Reads	15,000 per hour	15,000 per hour
Resource Manager API Writes	1,200 per hour	1,200 per hour
Resource Manager API request size	4,194,304 bytes	4,194,304 bytes
Tags per subscription ³	unlimited	unlimited
Unique tag calculations per subscription ³	10,000	10,000
Cloud services per subscription	Not Applicable ⁴	Not Applicable ⁴
Affinity groups per subscription	Not Applicable ⁴	Not Applicable ⁴

¹Default limits vary by offer Category Type, such as Free Trial, Pay-As-You-Go, and series, such as Dv2, F, G, etc.

²This includes both Standard and Premium storage accounts. If you require more than 200 storage accounts, make a request through [Azure Support](#). The Azure Storage team will review your business case and may approve up to 250 storage accounts.

³You can apply an unlimited number of tags per subscription. The number of tags per resource or resource group is limited to 15. Resource Manager only returns a [list of unique tag name and values](#) in the subscription when the number of tags is 10,000 or less. However, you can still find a resource by tag when the number exceeds 10,000.

⁴These features are no longer required with Azure Resource Groups and the Azure Resource Manager.

NOTE

It is important to emphasize that virtual machine cores have a regional total limit as well as a regional per size series (Dv2, F, etc.) limit that 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 would be allowed to deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores (for example, 10 A1 VMs and 20 D1 VMs).

Resource Group limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Resources per resource group (per resource type)	800	Varies per resource type
Deployments per resource group in the deployment history	800	800
Resources per deployment	800	800
Management Locks (per unique scope)	20	20
Number of Tags (per resource or resource group)	15	15

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Tag key length	512	512
Tag value length	256	256

Template limits

VALUE	DEFAULT LIMIT	MAXIMUM LIMIT
Parameters	256	256
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	1 MB	1 MB
Parameter file size	64 KB	64 KB

You can exceed some template limits by using a nested template. For more information, see [Using linked templates when deploying 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](#).

If you reach the limit of 800 deployments per resource group, delete deployments from the history that are no longer needed. You can delete entries from the history with [az group deployment delete](#) for Azure CLI, or [Remove-AzureRmResourceGroupDeployment](#) in PowerShell. Deleting an entry from the deployment history does not affect the deployed resources.

Virtual Machines limits

Virtual Machine 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 in Service Management (instead of Resource Manager) are automatically stored in a cloud service. You can add more virtual machines to that cloud service for load balancing and availability. See [How to Connect Virtual Machines with a Virtual Network or Cloud Service](#).

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. See [How to Set Up Endpoints to a Virtual Machine](#).

Virtual Machines limits - Azure Resource Manager

The following limits apply when using the Azure Resource Manager and Azure Resource Groups. Limits that have not changed with the Azure Resource Manager are not listed below. Please refer to the previous table for those

limits.

RESOURCE	DEFAULT LIMIT
Virtual machines per availability set	200
Certificates per subscription	Unlimited ¹

¹With Azure Resource Manager, certificates are stored in the Azure Key Vault. Although the number of certificates is unlimited for a subscription, there is still a 1 MB limit of certificates per deployment (which consists of either a single VM or an availability set).

Virtual Machine Scale Sets limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum number of VMs in a scale set	1000	1000
Maximum number of VMs based on a custom VM image in a scale set	300	300
Maximum number of scale sets in a region	2000	2000

Container Instances limits

RESOURCE	DEFAULT LIMIT
Container groups per subscription	20 ¹
Number of containers per container group	60
Number of volumes per container group	20
Ports per IP	5
Container creates per hour	60 ¹
Container creates per 5 minutes	20 ¹
Container deletes per hour	150 ¹
Container deletes per 5 minutes	50 ¹
Multiple containers per container group	Linux only ²
Azure Files volumes	Linux only ²
GitRepo volumes	Linux only ²
Secret volumes	Linux only ²

¹ Create an [Azure support request](#) to request a limit increase.

² Windows support for this feature is planned.

Container Registry limits

The following table details the features and limits of the Basic, Standard, and Premium [service tiers](#).

RESOURCE	BASIC	STANDARD	PREMIUM
Storage	10 GiB	100 GiB	500 GiB
ReadOps per minute ^{1, 2}	1k	300k	10,000k
WriteOps per minute ^{1, 3}	100	500	2k
Download bandwidth MBps ¹	30	60	100
Upload bandwidth MBps ¹	10	20	50
Webhooks	2	10	100
Geo-replication	N/A	N/A	Supported (preview)

¹ *ReadOps*, *WriteOps*, and *Bandwidth* are minimum estimates. ACR strives to improve performance as usage requires.

² [docker pull](#) translates to multiple read operations based on the number of layers in the image, plus the manifest retrieval.

³ [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.

Networking limits

ExpressRoute Limits

The following limits apply to ExpressRoute resources per subscription.

RESOURCE	DEFAULT LIMIT
ExpressRoute circuits per subscription	10
ExpressRoute circuits per region per subscription for ARM	10
Maximum number of routes for Azure private peering with ExpressRoute standard	4,000
Maximum number of routes for Azure private peering with ExpressRoute premium add-on	10,000
Maximum number of routes for Azure public peering with ExpressRoute standard	200
Maximum number of routes for Azure public peering with ExpressRoute premium add-on	200
Maximum number of routes for Azure Microsoft peering with ExpressRoute standard	200
Maximum number of routes for Azure Microsoft peering with ExpressRoute premium add-on	200

RESOURCE	DEFAULT LIMIT
Number of virtual network links allowed per ExpressRoute circuit	see table below

Number of Virtual Networks per ExpressRoute circuit

CIRCUIT SIZE	NUMBER OF VNET LINKS FOR STANDARD	NUMBER OF VNET 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

Networking limits

The following limits apply only for networking resources managed through the classic deployment model per subscription.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Virtual networks	50	100
Local network sites	20	contact support
DNS Servers per virtual network	20	100
Private IP Addresses per virtual network	4096	4096
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500K	500K
Network Security Groups (NSG)	100	200
NSG rules per NSG	200	400
User defined route tables	100	200
User defined routes per route table	100	400
Public IP addresses (dynamic)	5	contact support
Reserved public IP addresses	20	contact support

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Public VIP per deployment	5	contact support
Private VIP (ILB) per deployment	1	1
Endpoint Access Control Lists (ACLs)	50	50

Networking Limits - Azure Resource Manager

The following limits apply only for networking resources managed through Azure Resource Manager per region per subscription.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Virtual networks	50	1000
Subnets per virtual network	1000	10000
Virtual network peerings per Virtual Network	10	50
DNS Servers per virtual network	9	25
Private IP Addresses per virtual network	4096	8192
Private IP Addresses per network interface	256	1024
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500K	500K
Network Interfaces (NIC)	350	20000
Network Security Groups (NSG)	100	5000
NSG rules per NSG	200	500
IP addresses and ranges specified for source or destination in a security group	2000	4000
Application security groups	200	500
Application security groups per IP configuration, per NIC	10	20
IP configurations per application security group	1000	4000
Application security groups that can be specified within all security rules of a network security group	50	100
User defined route tables	100	200

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
User defined routes per route table	100	400
Public IP addresses - dynamic	(Basic) 60	contact support
Public IP addresses - static	(Basic) 20	contact support
Public IP addresses - static	(Standard) 20	contact support
Point-to-Site Root Certificates per VPN Gateway	20	20

Load Balancer limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Load Balancers	100	1000
Rules per resource, Basic	150	250
Rules per resource, Standard	1250	1500
Rules per IP configuration	299	299
Frontend IP configurations, Basic	10	contact support
Frontend IP configurations, Standard	10	600
Backend pool, Basic	100, single Availability Set	-
Backend pool, Standard	1000, single VNet	contact support
HA Ports, Standard	1 per internal frontend	-

[Contact support](#) in case you need to increase limits from default.

Application Gateway limits

RESOURCE	DEFAULT LIMIT	NOTE
Application Gateway	50 per subscription	Maximum 100
Frontend IP Configurations	2	1 public and 1 private
Frontend Ports	20	
Backend Address Pools	20	
Backend Servers per pool	100	
HTTP Listeners	20	
HTTP load balancing rules	200	# of HTTP Listeners * n, n=10 Default

RESOURCE	DEFAULT LIMIT	NOTE
Backend HTTP settings	20	1 per Backend Address Pool
Instances per gateway	10	For more instances, open support ticket
SSL certificates	20	1 per HTTP Listeners
Authentication certificates	5	Maximum 10
Request time out min	1 second	
Request time out max	24 hrs	
Number of sites	20	1 per HTTP Listeners
URL Maps per listener	1	
Maximum file upload size Standard	2 GB	
Maximum file upload size WAF	100 MB	

Network Watcher limits

RESOURCE	DEFAULT LIMIT	NOTE
Network Watcher	1 per region	
Packet Capture sessions	10 per region	# of sessions only, not saved captures

Traffic Manager limits

RESOURCE	DEFAULT LIMIT
Profiles per subscription	100 ¹
Endpoints per profile	200

¹Contact support in case you need to increase these limits.

DNS limits

RESOURCE	DEFAULT LIMIT
Zones per subscription	100 ¹
Record sets per zone	5000 ¹
Records per record set	20

¹ Contact Azure Support in case you need to increase these limits.

Storage limits

For additional details on storage account limits, see [Azure Storage Scalability and Performance Targets](#).

RESOURCE	DEFAULT LIMIT
Number of storage accounts per subscription	200 ¹
Max storage account capacity	500 TiB ²
Max number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account	No limit
Maximum request rate per storage account	20,000 requests per second ²
Max ingress ³ per storage account (US Regions)	10 Gbps if GRS/ZRS ⁴ enabled, 20 Gbps for LRS ²
Max egress ³ per storage account (US Regions)	20 Gbps if RA-GRS/GRS/ZRS ⁴ enabled, 30 Gbps for LRS ²
Max ingress ³ per storage account (Non-US regions)	5 Gbps if GRS/ZRS ⁴ enabled, 10 Gbps for LRS ²
Max egress ³ per storage account (Non-US regions)	10 Gbps if RA-GRS/GRS/ZRS ⁴ enabled, 15 Gbps for LRS ²

¹Includes both Standard and Premium storage accounts. If you require more than 200 storage accounts, make a request through [Azure Support](#). The Azure Storage team will review your business case and may approve up to 250 storage accounts.

² To get your standard storage accounts to grow past the advertised limits in capacity, ingress/egress and request rate, please make a request through [Azure Support](#). The Azure Storage team will review the request and may approve higher limits on a case by case basis.

³ Capped only by the account's ingress/egress limits. *Ingress* refers to all data (requests) being sent to a storage account. *Egress* refers to all data (responses) being received from a storage account.

⁴Azure Storage redundancy options include:

- **RA-GRS:** Read-access geo-redundant storage. If RA-GRS is enabled, egress targets for the secondary location are identical to those for the primary location.
- **GRS:** Geo-redundant storage.
- **ZRS:** Zone-redundant storage. Available only for block blobs.
- **LRS:** Locally redundant storage.

The following limits apply when performing management operations using the Azure Resource Manager only.

RESOURCE	DEFAULT LIMIT
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	200 per hour
Storage account management operations (list)	100 per 5 minutes

Azure Blob storage limits

RESOURCE	TARGET
Max size of single blob container	500 TiB

RESOURCE	TARGET
Max number of blocks in a block blob or append blob	50,000 blocks
Max size of a block in a block blob	100 MiB
Max size of a block blob	50,000 X 100 MiB (approx. 4.75 TiB)
Max size of a block in an append blob	4 MiB
Max size of an append blob	50,000 x 4 MiB (approx. 195 GiB)
Max size of a page blob	8 TiB
Max number of stored access policies per blob container	5
Target throughput for single blob	Up to 60 MiB per second, or up to 500 requests per second

Azure Files limits

For additional details on Azure Files limits, see [Azure Files scalability and performance targets](#).

RESOURCE	TARGET
Max size of a file share	5 TiB
Max size of a file in a file share	1 TiB
Max number of files in a file share	No limit
Max IOPS per share	1000 IOPS
Max number of stored access policies per file share	5
Maximum request rate per storage account	20,000 requests per second for files of any valid size ³
Target throughput for single file share	Up to 60 MiB per second
Maximum open handles for per file	2000 open handles
Maximum number of share snapshots	200 share snapshots

Azure File Sync limits

RESOURCE	TARGET	HARD LIMIT
Storage Sync Services per subscription	15 Storage Sync Services	No
Sync groups per Storage Sync Service	30 sync groups	Yes
Cloud endpoints per Sync Group	1 cloud endpoint	Yes
Server endpoints per Sync Group	50 server endpoints	No

RESOURCE	TARGET	HARD LIMIT
Server endpoints per server	33-99 server endpoints	Yes, but varies based on configuration
Endpoint size	4 TiB	No
File system objects (directories and files) per sync group	6 million objects	No
File size	100 GiB	No
Minimum file size for a file to be tiered	64 KiB	Yes

Azure Queue storage limits

RESOURCE	TARGET
Max size of single queue	500 TiB
Max size of a message in a queue	64 KiB
Max number of stored access policies per queue	5
Maximum request rate per storage account	20,000 messages per assuming 1 KiB message size
Target throughput for single queue (1 KiB messages)	Up to 2000 messages per second

Azure Table storage limits

RESOURCE	TARGET
Max size of single table	500 TiB
Max size of a table entity	1 MiB
Max number of properties in a table entity	252
Max number of stored access policies per table	5
Maximum request rate per storage account	20,000 transactions per second (assuming 1 KiB entity size)
Target throughput for single table partition (1 KiB entities)	Up to 2000 entities per second

Virtual machine disk limits

An Azure virtual machine supports attaching a number of data disks. This article describes scalability and performance targets for a VM's data disks. Use these targets to help decide 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 are not highly utilized at the same time, then the virtual machine can support a larger number of disks.

- **For Azure Managed Disks:** The disk limit for managed disks is per region and per disk type. The maximum

limit, and also the default limit, is 10,000 managed disks per region and per disk type for a subscription. For example, you can create up to 10,000 standard managed disks and also 10,000 premium managed disks in a region, per subscription.

Managed snapshots and images count against the managed disks limit.

- 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 (20,000/300 IOPS per disk), and for a Standard Tier VM, it is about 40 (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.

See [Virtual machine sizes](#) for additional details.

Managed virtual machine disks

Standard managed virtual machine disks

STANDARD DISK TYPE	S4	S6	S10	S20	S30	S40	S50
Disk size	32 GB	64 GB	128 GB	512 GB	1024 GB (1 TB)	2048 GB (2TB)	4095 GB (4 TB)
IOPS per disk	500	500	500	500	500	500	500
Throughput per disk	60 MB/sec	60 MB/sec	60 MB/sec				

Premium managed virtual machine disks: per disk limits

PREMIUM DISKS TYPE	P4	P6	P10	P20	P30	P40	P50
Disk size	32 GB	64 GB	128 GB	512 GB	1024 GB (1 TB)	2048 GB (2TB)	4095 GB (4 TB)
IOPS per disk	120	240	500	2300	5000	7500	7500
Throughput per disk	25 MB/sec	50 MB/sec	100 MB/sec	150 MB/sec	200 MB/sec	250 MB/sec	250 MB/sec

Premium managed virtual machine disks: per VM limits

RESOURCE	DEFAULT LIMIT
Max IOPS Per VM	80,000 IOPS with GS5 VM
Max 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	4095 GB	4095 GB
Max 8 KB IOPS per persistent disk	300	500
Max number of disks performing max 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
Max bandwidth per account (ingress + egress ¹)	<=50 Gbps

¹Ingress refers to all data (requests) being sent to a storage account. Egress refers to all data (responses) being 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	1024 GiB (1 TB)	2048 GiB (2 TB)	4095 GiB (4 TB)
Max IOPS per disk	500	2300	5000	7500	7500
Max throughput per disk	100 MB/s	150 MB/s	200 MB/s	250 MB/s	250 MB/s
Max number of disks per storage account	280	70	35	17	8

Premium unmanaged virtual machine disks: per VM limits

RESOURCE	DEFAULT LIMIT
Max IOPS Per VM	80,000 IOPS with GS5 VM
Max throughput per VM	2,000 MB/s with GS5 VM

Cloud Services limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Web/worker roles per deployment ¹	25	25

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Instance Input Endpoints per deployment	25	25
Input Endpoints per deployment	25	25
Internal Endpoints per deployment	25	25

¹Each Cloud Service with Web/Worker roles can have two deployments, one for production and one for staging. Also note that this limit refers to the number of distinct roles (configuration) and not the number of instances per role (scaling).

App Service limits

The following App Service limits include limits for Web Apps, Mobile Apps, API Apps, and Logic Apps.

RESOURCE	FREE	SHARED (PREVIEW)	BASIC	STANDARD	PREMIUM (PREVIEW)
Web, mobile, or API apps per App Service plan ¹	10	100	Unlimited ²	Unlimited ²	Unlimited ²
Logic apps per App Service plan ¹	10	10	10	20 per core	20 per core
App Service plan	1 per region	10 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
Scale-Out (max instances)	1 shared	1 shared	3 dedicated ³	10 dedicated ³	20 dedicated (50 in ASE) ^{3,4}
Storage ⁵	1 GB ⁵	1 GB ⁵	10 GB ⁵	50 GB ⁵	500 GB ^{4,5}
CPU time (5 min) ⁶	3 minutes	3 minutes	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates
CPU time (day) ⁶	60 minutes	240 minutes	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates
Memory (1 hour)	1024 MB per App Service plan	1024 MB per app	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
Application architecture	32-bit	32-bit	32-bit/64-bit	32-bit/64-bit	32-bit/64-bit
Web Sockets per instance ⁷	5	35	350	Unlimited	Unlimited

RESOURCE	FREE	SHARED (PREVIEW)	BASIC	STANDARD	PREMIUM (PREVIEW)
Concurrent debugger connections per application	1	1	1	5	5
azurewebsites.net subdomain with FTP/S and SSL	X	X	X	X	X
Custom domain support		X	X	X	X
Custom domain SSL support			Unlimited SNI SSL connections	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included
Integrated Load Balancer		X	X	X	X
Always On			X	X	X
Scheduled Backups				Scheduled backups every 2 hours, a max of 12 backups per day (manual + scheduled)	Scheduled backups every hour, a max of 50 backups per day (manual + scheduled)
Auto Scale				X	X
WebJobs ⁸	X	X	X	X	X
Azure Scheduler support		X	X	X	X
Endpoint monitoring			X	X	X
Staging Slots				5	20
Custom domains per app		500	500	500	500
SLA			99.9%	99.95% ¹⁰	99.95% ⁹

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. See [App Service Pricing](#) for more details.

⁴Premium tier allows up to 50 compute instances (subject to availability) and 500 GB of disk space when using App Service Environments, and 20 compute instances and 250 GB storage otherwise.

⁵The storage limit is the total content size across all apps in the same App Service plan. More storage options are

available in [App Service Environment](#)

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.

⁸Run 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. Azure Scheduler Free or Standard is required for scheduled WebJobs. There is no predefined limit on the number of WebJobs that can run in an App Service instance, but there are practical limits that depend on what the application code is trying to do.

⁹SLA of 99.95% provided for deployments that use multiple instances with Azure Traffic Manager configured for failover.

Scheduler limits

The following table describes each of the major quotas, limits, defaults, and throttles in Azure Scheduler.

RESOURCE	LIMIT DESCRIPTION
Job size	Maximum job size is 16K. If a PUT or a PATCH results in a job larger than these limits, a 400 Bad Request status code is returned.
Request URL size	Maximum size of the request URL is 2048 chars.
Aggregate header size	Maximum aggregate header size is 4096 chars.
Header count	Maximum header count is 50 headers.
Body size	Maximum body size is 8192 chars.
Recurrence span	Maximum recurrence span is 18 months.
Time to start time	Maximum "time to start time" is 18 months.
Job history	Maximum response body stored in job history is 2048 bytes.
Frequency	The default max frequency quota is 1 hour in a free job collection and 1 minute in a standard job collection. The max frequency is configurable on a job collection to be lower than the maximum. All jobs in the job collection are limited the value set on the job collection. If you attempt to create a job with a higher frequency than the maximum frequency on the job collection then request will fail with a 409 Conflict status code.
Jobs	The default max jobs quota is 5 jobs in a free job collection and 50 jobs in a standard job collection. The maximum number of jobs is configurable on a job collection. All jobs in the job collection are limited the value set on the job collection. If you attempt to create more jobs than the maximum jobs quota, then the request fails with a 409 Conflict status code.
Job collections	Maximum number of job collection per subscription is 200,000.

RESOURCE	LIMIT DESCRIPTION
Job history retention	Job history is retained for up to 2 months or up to the last 1000 executions.
Completed and faulted job retention	Completed and faulted jobs are retained for 60 days.
Timeout	There's a static (not configurable) request timeout of 60 seconds for HTTP actions. For longer running operations, follow HTTP asynchronous protocols; for example, return a 202 immediately but continue working in the background.

Batch limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Batch accounts per region per subscription	3	50
Dedicated cores per Batch account	20	N/A ¹
Low-priority cores per Batch account	20	N/A ²
Active jobs and job schedules ³ per Batch account	20	5000 ⁴
Pools per Batch account	20	2500

¹ The number of dedicated cores per Batch account can be increased, but the maximum number is unspecified. Contact Azure support to discuss increase options.

² The number of low-priority cores per Batch account can be increased, but the maximum number is unspecified. Contact Azure support to discuss increase options.

³ Completed jobs and job schedules are not limited.

⁴ Contact Azure support if you want to request an increase beyond this limit.

BizTalk Services limits

The following table shows the limits for Azure Biztalk Services.

RESOURCE	FREE (PREVIEW)	DEVELOPER	BASIC	STANDARD	PREMIUM
Scale out	N/A	N/A	Yes, in increments of 1 Basic Unit	Yes, in increments of 1 Standard Unit	Yes, in increments of 1 Premium Unit
Scale Limit	N/A	N/A	Up to 8 units	Up to 8 units	Up to 8 units
EAI Bridges per Unit	N/A	25	25	125	500
EDI Agreements per Unit	N/A	10	50	250	1000

RESOURCE	FREE (PREVIEW)	DEVELOPER	BASIC	STANDARD	PREMIUM
Hybrid Connections per Unit	5	5	10	50	100
Hybrid Connection Data Transfer (GBs) per Unit	5	5	50	250	500
Number of connections using BizTalk Adapter Service per Unit	N/A	1	2	5	25
Archiving	N/A	Available	N/A	N/A	Available
High Availability	N/A	N/A	Available	Available	Available

Azure Cosmos DB limits

Azure Cosmos DB is a global scale database in which throughput and storage can be scaled to handle whatever your application requires. If you have any questions about the scale Azure Cosmos DB provides, please send email to askcosmosdb@microsoft.com.

Mobile Engagement limits

RESOURCE	MAXIMUM LIMIT
App Collection Users	5 per App Collection
Average Data points	200 per Active User/Day
Average App-Info set	50 per Active User/Day
Average Messages pushed	20 per Active User/Day
Segments	100 per app
Criteria per segment	10
Active Push Campaigns	50 per app
Total Push Campaigns (includes Active & Completed)	1000 per app

Search limits

Pricing tiers determine the capacity and limits of your search service. Tiers include:

- *Free* multi-tenant service, shared with other Azure subscribers, 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 (S1, S2, S3, 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 provisioned at a specific tier, 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 a SKU or tier for Azure Search](#).

Maximum service limits can be raised upon request. Contact Azure Support if you need more services within the same subscription.

RESOURCE	FREE	BASIC	S1	S2	S3	S3 HD ¹
Maximum services	1	12	12	6	6	6
Maximum scale in SU ²	N/A ³	3 SU ⁴	36 SU	36 SU	36 SU	36 SU

¹ S3 HD does not support [indexers](#) at this time.

² Search units (SU) 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 how search units are computed, plus a chart of valid combinations that stay under the maximum limits, see [Scale resource levels for query and index workloads](#).

³ Free is based on shared resources used by multiple subscribers. At this tier, there are no dedicated resources for an individual subscriber. For this reason, maximum scale is marked as not applicable.

⁴ Basic has one fixed partition. At this tier, additional SUs are used for allocating more replicas for increased query workloads.

Limits per search service

Storage is constrained by disk space or by a hard limit on the *maximum number* of indexes or documents, whichever comes first.

RESOURCE	FREE	BASIC	S1	S2	S3	S3 HD
Service Level Agreement (SLA)	No ¹	Yes	Yes	Yes	Yes	Yes
Storage per partition	50 MB	2 GB	25 GB	100 GB	200 GB	200 GB
Partitions per service	N/A	1	12	12	12	3 ²
Partition size	N/A	2 GB	25 GB	100 GB	200 GB	200 GB
Replicas	N/A	3	12	12	12	12
Maximum indexes	3	5	50	200	200	1000 per partition or 3000 per service

Resource	Free	Basic	S1	S2	S3	S3 HD
Maximum indexers	3	5	50	200	200	No indexer support
Maximum datasources	3	5	50	200	200	No indexer support
Maximum documents	10,000	1 million	15 million per partition or 180 million per service	60 million per partition or 720 million per service	120 million per partition or 1.4 billion per service	1 million per index or 200 million per partition

¹ Free tier and preview features do not come with service level agreements (SLAs). For all billable tiers, SLAs take effect when you provision sufficient redundancy for your service. Two or more replicas are required for query (read) SLA. Three or more replicas are required for query and indexing (read-write) SLA. The number of partitions is not an SLA consideration.

² S3 HD has a hard limit of 3 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.

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 Search](#).

Media Services limits

NOTE

For resources that are not fixed, you may ask for the quotas to be raised, by opening a support ticket. Do **not** create additional Azure Media Services accounts in an attempt to obtain higher limits.

Resource	Default Limit
Azure Media Services (AMS) accounts in a single subscription	25 (fixed)
Media Reserved Units (RUs) per AMS account	25 (S1, S2) 10 (S3) ⁽¹⁾
Jobs per AMS account	50,000 ⁽²⁾
Chained tasks per job	30 (fixed)
Assets per AMS 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 AMS account	5

RESOURCE	DEFAULT LIMIT
Programs in stopped state per channel	50
Programs in running state per channel	3
Streaming endpoints in running state per AMS 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 is a limit on the maximum file size supported for processing in Media Services. ⁷

¹ S3 RUs are not available in India West. If you change the type (for example, from S2 to S1,) the max RU limits are reset.

² This number includes queued, finished, active, and canceled jobs. It does not include deleted jobs. You can delete the old jobs using **IJob.Delete** or the **DELETE** HTTP request.

As of April 1, 2017, any Job record in your account older than 90 days will be automatically deleted, along with its associated Task records, even if the total number of records is below the maximum quota. If you need to archive the job/task information, you can use the code described [here](#).

³ When making a request to list Job entities, a maximum of 1,000 jobs is returned per request. If you need to keep track of all submitted Jobs, you can use top/skip as described in [OData system query options](#).

⁴ Locators are not 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 [this](#) section.

⁵ The storage accounts must be from the same Azure subscription.

⁶ There is a limit of 1,000,000 policies for different AMS policies (for example, for Locator policy or ContentKeyAuthorizationPolicy).

NOTE

You should use the same policy ID if you are always using the same days / access permissions / etc. For information and an example, see [this](#) section.

⁷If you are uploading content to an Asset in Azure Media Services to process it with one of the media processors in the service (that is, encoders like Media Encoder Standard and Media Encoder Premium Workflow, or analysis engines like Face Detector), then you should be aware of the constraints on the maximum file sizes supported.

The maximum size supported for a single blob is currently up to 5 TB in Azure Blob Storage. However, additional limits apply in Azure Media Services based on the VM sizes that are used by the service. The following table shows the limits on each of the Media Reserved Units (S1, S2, S3.) If your source file is larger than the limits defined in the table, your encoding job will fail. If you are encoding 4K resolution sources of long duration, you are required to use S3 Media Reserved Units to achieve the performance needed. If you have 4K content that is larger than 260 GB limit on the S3 Media Reserved Units, contact us at amshelp@microsoft.com for potential mitigations to support your scenario.

MEDIA RESERVED UNIT TYPE	MAXIMUM INPUT SIZE (GB)
S1	325
S2	640
S3	260

CDN limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
CDN profiles	25	25
CDN endpoints per profile	10	25
Custom domains per endpoint	10	25

A CDN subscription can contain one or more CDN profiles and a CDN profile can contain one or more CDN endpoints. You may wish to use multiple profiles to organize your CDN endpoints by internet domain, web application, or some other criteria.

To request an update to your subscription's default limits, open a support ticket.

Mobile Services limits

TIER:	FREE	BASIC	STANDARD
API Calls	500 K	1.5 M / unit	15 M / unit
Active Devices	500	Unlimited	Unlimited
Scale	N/A	Up to 6 units	Unlimited units
Push Notifications	Notification Hubs Free Tier included, up to 1 M pushes	Notification Hubs Basic Tier included, up to 10 M pushes	Notification Hubs Standard Tier included, up to 10 M pushes
Real time messaging/ Web Sockets	Limited	350 / mobile service	Unlimited
Offline synchronizations	Limited	Included	Included
Scheduled jobs	Limited	Included	Included
SQL Database (required) Standard rates apply for additional capacity	20 MB included	20 MB included	20 MB included
CPU capacity	60 minutes / day	Unlimited	Unlimited
Outbound data transfer	165 MB per day (daily Rollover)	Included	Included

For additional details on these limits and for information on pricing, see [Mobile Services Pricing](#).

Monitor limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Autoscale Settings	100 per region per subscription	same as default
Metric Alerts	100 active alert rules per subscription	call support
Near-Real Time Alerts (Preview)	20 active alert rules per subscription	same as default during preview

Notification Hub Service limits

TIER:	FREE	BASIC	STANDARD
Included Pushes	1 Million	10 Million	10 Million
Active Devices	500	200,000	10 million
Tag quota per installation/registration	60	60	60

For additional details on these limits and for information on pricing, see [Notification Hubs Pricing](#).

Event Hubs limits

The following table lists quotas and limits specific to [Azure Event Hubs](#). For information about Event Hubs pricing, see [Event Hubs pricing](#).

LIMIT	SCOPE	TYPE	BEHAVIOR WHEN EXCEEDED	VALUE
Number of event hubs per namespace	Namespace	Static	Subsequent requests for creation of a new event hub will be rejected.	10
Number of partitions per event hub	Entity	Static	-	32
Number of consumer groups per event hub	Entity	Static	-	20
Number of AMQP connections per namespace	Namespace	Static	Subsequent requests for additional connections will be rejected and an exception is received by the calling code.	5,000
Maximum size of Event Hubs event	System-wide	Static	-	256 KB
Maximum size of an event hub name	Entity	Static	-	50 characters

LIMIT	SCOPE	TYPE	BEHAVIOR WHEN EXCEEDED	VALUE
Number of non-epoch receivers per consumer group	Entity	Static	-	5
Maximum retention period of event data	Entity	Static	-	1-7 days
Maximum throughput units	Namespace	Static	Exceeding the throughput unit limit causes your data to be throttled and generates a ServerBusyException . You can request a larger number of throughput units for a Standard tier by filing 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	Static	Subsequent requests for authorization rule creation will be rejected.	12

Service Bus limits

The following table lists quota information specific to Service Bus messaging. For information about pricing and other quotas for Service Bus, see the [Service Bus Pricing](#) overview.

QUOTA NAME	SCOPE	TYPE	BEHAVIOR WHEN EXCEEDED	VALUE
Maximum number of basic / standard namespaces per Azure subscription	Namespace	Static	Subsequent requests for additional basic / standard namespaces will be rejected by the portal.	100
Maximum number of premium namespaces per Azure subscription	Namespace	Static	Subsequent requests for additional premium namespaces will be rejected by the portal.	10
Queue/topic size	Entity	Defined upon creation of the queue/topic.	Incoming messages will be rejected and an exception will be received by the calling code.	1, 2, 3, 4 or 5 GB. If partitioning is enabled, the maximum queue/topic size is 80 GB.

Quota Name	Scope	Type	Behavior When Exceeded	Value
Number of concurrent connections on a namespace	Namespace	Static	Subsequent requests for additional connections will be rejected and an exception will be received by the calling code. REST operations do not count towards concurrent TCP connections.	NetMessaging: 1,000 AMQP: 5,000
Number of concurrent receive requests on a queue/topic/subscription entity	Entity	Static	Subsequent receive requests will be rejected and an exception will be 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/queues per service namespace	System-wide	Static	Subsequent requests for creation of a new topic or queue on the service namespace will be rejected. As a result, if configured through the Azure portal , an error message will be generated. If called from the management API, an exception will be received by the calling code.	10,000 The total number of topics plus queues in a service namespace must be less than or equal to 10,000. This is not applicable to Premium as all entities are partitioned.
Number of partitioned topics/queues per service namespace	System-wide	Static	Subsequent requests for creation of a new partitioned topic or queue on the service namespace will be rejected. As a result, if configured through the Azure portal , an error message will be generated. If called from the management API, a QuotaExceededException exception will be received by the calling code.	Basic and Standard Tiers - 100 Premium - 1,000 (per messaging unit) Each partitioned queue or topic counts towards the quota of 10,000 entities per namespace.

Quota Name	Scope	Type	Behavior When Exceeded	Value
Maximum size of any messaging entity path: queue or topic	Entity	Static	-	260 characters
Maximum size of any messaging entity name: namespace, subscription, or subscription rule	Entity	Static	-	50 characters
Message size for a queue/topic/subscription entity	System-wide	Static	Incoming messages that exceed these quotas will be rejected and an exception will be received by the calling code.	<p>Maximum message size: 256KB (Standard tier) / 1MB (Premium tier).</p> <p>Note Due to system overhead, this limit is usually slightly less.</p> <p>Maximum header size: 64KB</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/subscription entity	System-wide	Static	A SerializationException exception is generated.	Maximum message property size for each property is 32K. Cumulative size of all properties cannot exceed 64K. This applies to the entire header of the BrokeredMessage , which has both user properties as well as system properties (such as SequenceNumber , Label , MessageId , and so on).

Quota Name	Scope	Type	Behavior When Exceeded	Value
Number of subscriptions per topic	System-wide	Static	Subsequent requests for creating additional subscriptions for the topic will be rejected. As a result, if configured through the portal, an error message will be shown. If called from the management API an exception will be received by the calling code.	2,000
Number of SQL filters per topic	System-wide	Static	Subsequent requests for creation of additional filters on the topic will be rejected and an exception will be received by the calling code.	2,000
Number of correlation filters per topic	System-wide	Static	Subsequent requests for creation of additional filters on the topic will be rejected and an exception will be received by the calling code.	100,000
Size of SQL filters/actions	System-wide	Static	Subsequent requests for creation of additional filters will be rejected and an exception will be received by the calling code.	<p>Maximum length of filter condition string: 1024 (1K).</p> <p>Maximum length of rule action string: 1024 (1K).</p> <p>Maximum number of expressions per rule action: 32.</p>
Number of SharedAccessAuthorizationRule rules per namespace, queue, or topic	Entity, namespace	Static	Subsequent requests for creation of additional rules will be rejected and an exception will be received by the calling code.	<p>Maximum number of rules: 12.</p> <p>Rules that are configured on a Service Bus namespace apply to all queues and topics in that namespace.</p>

IoT Hub limits

The following table lists the limits associated with the different service tiers (S1, S2, S3, F1). For information about the cost of each *unit* in each tier, see [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 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	10
Maximum free IoT hubs per Azure subscription	1
Maximum number of device identities returned in a single call	1000
IoT Hub message maximum retention for device-to-cloud messages	7 days
Maximum size of device-to-cloud message	256 KB
Maximum size of device-to-cloud batch	256 KB
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 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 (tags, reported properties, and desired properties)	8 KB
Maximum size of device twin string value	512 bytes
Maximum depth of object in device twin	5
Maximum size of direct method payload	8 KB
Job history maximum retention	30 days

RESOURCE	LIMIT
Maximum concurrent jobs	10 (for S3), 5 for (S2), 1 (for S1)
Maximum additional endpoints	10 (for S1, S2, S3)
Maximum message routing rules	100 (for S1, S2, S3)

NOTE

If you need more than 10 paid IoT hubs in an Azure subscription, contact Microsoft support.

NOTE

Currently, the maximum number of devices you can connect to a single IoT hub is 500,000. If you want to increase this limit, contact [Microsoft Support](#).

The IoT Hub service throttles requests when the following quotas are exceeded:

THROTTLE	PER-HUB VALUE
Identity registry operations (create, retrieve, list, update, delete), individual or bulk import/export	83.33/sec/unit (5000/min/unit) (for S3) 1.67/sec/unit (100/min/unit) (for S1 and S2).
Device connections	6000/sec/unit (for S3), 120/sec/unit (for S2), 12/sec/unit (for S1). Minimum of 100/sec.
Device-to-cloud sends	6000/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 (5000/min/unit) (for S3), 1.67/sec/unit (100/min/unit) (for S1 and S2).
Cloud-to-device receives	833.33/sec/unit (50000/min/unit) (for S3), 16.67/sec/unit (1000/min/unit) (for S1 and S2).
File upload operations	83.33 file upload notifications/sec/unit (5000/min/unit) (for S3), 1.67 file upload notifications/sec/unit (100/min/unit) (for S1 and S2). 10000 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	3000/sec/unit (for S3), 60/sec/unit (for S2), 20/sec/unit (for S1)
Device twin reads	50/sec/unit (for S3), Maximum of 10/sec or 1/sec/unit (for S2), 10/sec (for S1)
Device twin updates	50/sec/unit (for S3), Maximum of 10/sec or 1/sec/unit (for S2), 10/sec (for S1)

THROTTLE	PER-HUB VALUE
Jobs operations (create, update, list, delete)	83.33/sec/unit (5000/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)

IoT Hub Device Provisioning Service limits

The following table lists the limits that apply to IoT Hub Device Provisioning Service resources:

RESOURCE	LIMIT
Maximum Device Provisioning Services per Azure subscription	10
Maximum number of enrollments	10,000
Maximum number of registrations	10,000
Maximum number of enrollment groups	100
Maximum number of CAs	10

NOTE

These limits are for public preview. Once the service is generally available, you can contact [Microsoft Support](#) to increase the number of instances in your subscription.

The Device Provisioning Service throttles requests when the following quotas are exceeded:

THROTTLE	PER-SERVICE VALUE
Operations	100/min
Device registrations	100/min

Data Factory limits

Data factory is a multi-tenant service that has the following default limits in place to make sure customer subscriptions are protected from each other's workloads. Many of the limits can be easily raised for your subscription up to the maximum limit by contacting support.

Version 2

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Data factories in an Azure subscription	50	Contact support
Pipelines within a data factory	2500	Contact support
Datasets within a data factory	2500	Contact support
Triggers within a data factory	2500	Contact support

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Linked services within a data factory	2500	Contact support
Integration runtimes within a data factory ⁴	2500	Contact support
Concurrent pipeline runs per pipeline	20	Contact support
Max activities per pipeline	20	30
Max parameters per pipeline	20	30
Bytes per object for pipeline objects ¹	200 KB	200 KB
Bytes per object for dataset and linked service objects ¹	100 KB	2000 KB
Cloud data movement units ³	32	Contact support
Retry count for pipeline activity runs	1 day(timeout)	1 day (timeout)
Write API calls	2500/hr This limit is imposed by Azure Resource Manager, not Azure Data Factory.	Contact support .
Read API calls	12,500/hr This limit is imposed by Azure Resource Manager, not Azure Data Factory.	Contact support

Version 1

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Data factories in an Azure subscription	50	Contact support
Pipelines within a data factory	2500	Contact support
Datasets within a data factory	5000	Contact support
Concurrent slices per dataset	10	10
Bytes per object for pipeline objects ¹	200 KB	200 KB
Bytes per object for dataset and linked service objects ¹	100 KB	2000 KB
HDInsight on-demand cluster cores within a subscription ²	60	Contact support
Cloud data movement units ³	32	Contact support

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Retry count for pipeline activity runs	1000	MaxInt (32 bit)

¹ Pipeline, dataset, and linked service objects represent a logical grouping of your workload. Limits for these objects do not relate to amount of data you can move and process with the Azure Data Factory service. 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 above limit is the Data Factory enforced core limit for on-demand HDInsight cores and is different from the core limit associated with your Azure subscription.

³ Cloud data movement unit (DMU) is being used in a cloud-to-cloud copy operation. It is a measure that represents the power (a combination of CPU, memory, and network resource allocation) of a single unit in Data Factory. You can achieve higher copy throughput by using more DMUs for some scenarios. Refer to [Cloud data movement units](#) section on details.

⁴ The Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory to provide the following data integration capabilities across different network environments: data movement, dispatching activities to compute services, execution of SSIS packages. For more information, see [Integration Runtime overview](#).

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

Data Lake Analytics makes the complex task of managing distributed infrastructure and complex code easy. It dynamically provisions resources and lets you do analytics on exabytes of data. When the job completes, it winds down resources automatically, and you pay only for the processing power used. As you increase or decrease the size of data stored or the amount of compute used, you don't have to rewrite code. Many of the default limits can be easily raised for your subscription by contacting 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.
Maximum script size for job submission	3 MB	

Data Lake Store limits

Azure Data Lake Store is an enterprise-wide hyper-scale repository for big data analytic workloads. Data Lake Store enables you to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics. There is no limit to the amount of data you can store in a Data Lake Store account.

RESOURCE	DEFAULT LIMIT	COMMENTS
Max number of Data Lake Store accounts, per subscription, per region	10	Contact Support to request an increase for this limit
Max number of access ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries
Max number of default ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries

Database Migration Service Limits

The 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	Contact Support to request an increase for this limit

Stream Analytics limits

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of Streaming Units per subscription per region	200	A request to increase streaming units for your subscription beyond 200 can be made by contacting Microsoft Support .
Maximum number of inputs per job	60	There is a hard limit of 60 inputs per Stream Analytics job.
Maximum number of outputs per job	60	There is a hard limit of 60 outputs per Stream Analytics job.
Maximum number of functions per job	60	There is a hard limit of 60 functions per Stream Analytics job.
Maximum number of Streaming Units per job	120	There is a hard limit of 120 Streaming Units per Stream Analytics job.
Maximum number of jobs per region	1500	Each subscription may have up to 1500 jobs per geographical region.
Reference data blob MB	100	Reference data blobs cannot be larger than 100 MB each.

Active Directory limits

Here are the usage constraints and other service limits for the Azure Active Directory service.

CATEGORY	LIMITS

CATEGORY	LIMITS
Directories	<p>A single user can only be associated with a maximum of 20 Azure Active Directory directories.</p> <p>Examples of possible combinations:</p> <ul style="list-style-type: none"> • A single user creates 20 directories. • A single user is added to 20 directories as a member. • A single user creates 10 directories and later is added by others to 10 different directories.
Objects	<ul style="list-style-type: none"> • A maximum of 500,000 objects can be created in a single directory by users of the Free edition of Azure Active Directory. • A non-admin user can create no more than 250 objects.
Schema extensions	<ul style="list-style-type: none"> • String type extensions can have maximum of 256 characters. • Binary type extensions are limited to 256 bytes. • 100 extension values (across ALL types and ALL applications) can be written to any single object. • 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 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.
Groups	<ul style="list-style-type: none"> • A maximum of 100 users can be owners of a single group. • Any number of objects can be members of a single group in Azure Active Directory. • The number of members in a group you can synchronize from your on-premises Active Directory to Azure Active Directory is limited to 15K members, using Azure Active Directory Directory Synchronization (DirSync). • The number of members in a group you can synchronize from your on-premises Active Directory to Azure Active Directory using Azure AD Connect is limited to 50K members.
Access Panel	<ul style="list-style-type: none"> • There is no limit to the number of applications that can be seen in the Access Panel per end user, for users assigned licenses for Azure AD Premium or the Enterprise Mobility Suite. • A maximum of 10 app tiles (examples: Box, Salesforce, or Dropbox) can be seen in the Access Panel for each end user for users assigned licenses for Free or Azure AD Basic editions of Azure Active Directory. This limit does not apply to Administrator accounts.

CATEGORY	LIMITS
Reports	A maximum of 1,000 rows can be viewed or downloaded in any report. Any additional data is truncated.
Administrative units	An object can be a member of no more than 30 administrative units.

Azure Event Grid limits

RESOURCE	LIMIT
Event Subscriptions per region	1000
Custom Topics per region	20

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 (24*7).
Maximum size of a tiered volume on physical devices	64 TB for 8100 and 8600	8100 and 8600 are physical devices.
Maximum size of a tiered volume on virtual devices in Azure	30 TB for 8010 64 TB for 8020	8010 and 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 8100 24 TB for 8600	8100 and 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	

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of schedules per backup policy	10	
Maximum number of snapshots of any type that can be retained per volume	256	This includes local snapshots and cloud snapshots.
Maximum number of snapshots that can be present in any device	10,000	
Maximum number of volumes that can be processed in parallel for backup, restore, or clone	16	<ul style="list-style-type: none"> If there are more than 16 volumes, they will be processed sequentially as processing slots become available. New backups of a cloned or a restored tiered volume cannot occur until the operation is finished. However, 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 restore or clone operation, regardless of the volume size. The volume performance may initially be slower than normal as most of the data and metadata still resides in the cloud. Performance may 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 may 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 cannot 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 may initially be slower than normal as most of the data and metadata still resides in the cloud. Performance may 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 may be affected by Internet bandwidth to the cloud. Unlike tiered volumes, in the case of 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 may 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/s with a single 10GbE network interface	Up to 2x with MPIO and two network interfaces.
Maximum client read/write throughput (when served from the HDD tier)*	120/250 MB/s	
Maximum client read/write throughput (when served from the cloud tier)*	11/41 MB/s	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 may be lower and depends on I/O mix and network conditions.

Log Analytics limits

NOTE

Log Analytics was formerly known as Operational Insights.

The following limits apply to Log Analytics resources per subscription:

RESOURCE	DEFAULT LIMIT	COMMENTS
Number of free workspaces per subscription	10	This limit cannot be increased.
Number of paid workspaces per subscription	N/A	You are limited by the number of resources within a resource group and number of resource groups per subscription

The following limits apply to each Log Analytics workspace:

	FREE	STANDARD	PREMIUM	STANDALONE	OMS
Data volume collected per day	500 MB ¹	None	None	None	None
Data retention period	7 days	1 month	12 months	1 month ²	1 month ²

¹ When customers reach their 500 MB daily data transfer limit, data analysis stops and resumes at the start of the next day. A day is based on UTC.

² The data retention period for the Standalone and OMS pricing plans can be increased to 730 days.

CATEGORY	LIMITS	COMMENTS
Data Collector API	Maximum size for a single post is 30 MB Maximum size for field values is 32 KB	Split larger volumes into multiple posts Fields longer than 32 KB are truncated.
Search API	5000 records returned for non-aggregated data 500000 records for aggregated data	Aggregated data is a search that includes the <code>measure</code> command

Backup limits

The following limits apply to Azure Backup.

LIMIT IDENTIFIER	DEFAULT LIMIT
Number of servers/machines that can be registered against each vault	50 for Windows Server/Client/SCDPM 200 for IaaS VMs
Size of a data source for data stored in Azure vault storage	54400 GB max ¹
Number of backup vaults that can be created in each Azure subscription	25 Recovery Services vaults per region

LIMIT IDENTIFIER	DEFAULT LIMIT
Number of times backup can be scheduled per day	3 per day for Windows Server/Client 2 per day for SCDPM Once a day for IaaS VMs
Data disks attached to an Azure virtual machine for backup	16
Size of individual data disk attached to an Azure virtual machine for backup	1023 GB ²

- ¹The 54400 GB limit does not apply to IaaS VM backup.
- ² We have a [private preview](#) for supporting unmanaged disks upto 4TB.

Site Recovery limits

The following limits apply to Azure Site Recovery:

LIMIT IDENTIFIER	DEFAULT LIMIT
Number of vaults per subscription	25
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

Application Insights limits

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, you can increase the limit up to 1,000 GB from the portal. For capacities greater than 1,000 GB, send mail to AIDataCap@microsoft.com .
Free data per month (Basic price plan)	1 GB	Additional data is charged per gigabyte.
Throttling	32 k 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.

RESOURCE	DEFAULT LIMIT	NOTE
Maximum event size	64 K	
Property and metric name length	150	See type schemas
Property value string length	8,192	See type schemas
Trace and exception message length	10 k	See type schemas
Availability tests count per app	100	
Profiler data retention	5 days	
Profiler data sent per day	10GB	

For more information, see [About pricing and quotas in Application Insights](#).

API Management limits

RESOURCE	LIMIT
API Calls (per unit of scale)	32 million per day ¹
Data transfer (per unit of scale)	161 GB per day ¹
Cache	5 GB ¹
Units of scale	Unlimited ¹
Azure Active Directory Integration	Unlimited User Accounts ¹

¹API Management limits are different for each pricing tier. To see the pricing tiers and their associated limits and scaling options, see [API Management Pricing](#).

Azure Redis Cache limits

RESOURCE	LIMIT
Cache size	530 GB
Databases	64
Max connected clients	40,000
Redis Cache replicas (for high availability)	1
Shards in a premium cache with clustering	10

Azure Redis Cache limits and sizes are different for each pricing tier. To see the pricing tiers and their associated sizes, see [Azure Redis Cache Pricing](#).

For more information on Azure Redis Cache configuration limits, see [Default Redis server configuration](#).

Because configuration and management of Azure Redis Cache instances is done by Microsoft, not all Redis

commands are supported in Azure Redis Cache. For more information, see [Redis commands not supported in Azure Redis Cache](#).

Key Vault limits

Key transactions (Max 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 2048-bit	5	1000	10	2000
RSA 3072-bit	5	250	10	500
RSA 4096-bit	5	125	10	250

Secrets, Managed Storage Account Keys, and vault transactions:

TRANSACTIONS TYPE	MAX TRANSACTIONS ALLOWED IN 10 SECONDS, PER VAULT PER REGION ¹
All transactions	2000

¹ There is a subscription-wide limit for all transaction types, that is 5x per key vault limit. For example, HSM- other transactions per subscription are limited to 5000 transactions in 10 seconds per subscription.

Multi-Factor Authentication

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Max number of Trusted IP addresses/ranges per subscription	0	50
Remember my devices - number of days	14	60
Max 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	1800
Lock user account after X consecutive MFA denials	Not Set	99
Reset account lockout counter after X minutes	Not Set	9999
Unlock account after X minutes	Not Set	9999

Automation limits

RESOURCE	MAXIMUM LIMIT
Max number of new jobs that can be submitted every 30 seconds per Automation Account (non Scheduled jobs)	100
Max number of concurrent running jobs at the same instance of time per Automation Account (non Scheduled jobs)	200
Max number of modules that can be imported every 30 seconds per Automation Account	5
Max size of a Module	100 MB
Job Run Time - Free tier	500 minutes per subscription per calendar month
Max amount of memory given to a job	400 MB
Max number of network sockets allowed per job	1000

SQL Database limits

For SQL Database limits, see [SQL Database Resource Limits](#).

SQL Data Warehouse limits

For SQL Data Warehouse limits, see [SQL Data Warehouse Resource Limits](#).

See also

[Understanding Azure Limits and Increases](#)

[Virtual Machine and Cloud Service Sizes for Azure](#)

[Sizes for Cloud Services](#)

Azure SDK for .NET 3.0 release notes

9/19/2017 • 2 min to read • [Edit Online](#)

This topic includes release notes for version 3.0 of the Azure SDK for .NET.

Azure SDK for .NET 3.0 release summary

Release date: 03/07/2017

No breaking changes to the Azure SDK 3.0 have been introduced in this release. There is also no upgrade process needed to leverage this SDK with existing Cloud Service projects. To allow use of the Azure SDK 3.0 without requiring an upgrade process, Azure SDK 3.0 installs to the same directories as Azure SDK 2.9. Most the components did not change the major version from 2.9 but instead just updated the build number.

Visual Studio 2017 RTW

- In Visual Studio 2017, this release of the Azure SDK for .NET is built in to the Azure Workload. All the tools you need to do Azure development will be part of Visual Studio 2017 going forward. For Visual Studio 2015 the SDK will still be available through WebPI. We have discontinued Azure SDK for .NET releases for Visual Studio 2013 now that Visual Studio 2017 has been released.

Azure Diagnostics

- Changed the behavior to only store a partial connection string with the key replaced by a token for Cloud Services diagnostics storage connection string. The actual storage key is now stored in the user profile folder so its access can be controlled. Visual Studio will read the storage key from user profile folder for local debugging and publishing process.
- In response to the change described above, Visual Studio Online team enhanced the Azure Cloud Services deployment task template so users could specify the storage key for setting diagnostics extension when publishing to Azure in Continuous Integration and Deployment.
- We've made it possible to store secure connection string and tokenization for Azure Diagnostics (WAD), to help you solve problems with configuration across environments.

Windows Server 2016 virtual machines

- Visual Studio now supports deploying Cloud Services to OS Family 5 (Windows Server 2016) virtual machines. For existing cloud services, you can change your settings to target the new OS Family. When creating new cloud services, if you choose to create the service using .net 4.6 or higher, it will default the service to use OS Family 5. For more information, you can review the [Guest OS Family support table](#).

Known issues

- Azure .NET SDK 3.0 introduced an issue when removing Visual Studio 2017 in a side by side configuration with Visual Studio 2015. If you have the Azure SDK installed for Visual Studio 2015, the Microsoft Azure Storage Emulator and Microsoft Azure Compute Emulator will be removed if you uninstall Visual Studio 2017. This will produce an error when creating and debugging new Cloud services projects in Visual Studio 2015. In order to fix this issue, reinstall the Azure SDK from the Web Platform Installer. The issue will be resolved in a future Visual Studio 2017 update..

Azure In-Role Cache

- Support for Azure In-Role Cache ended on November 30, 2016. For more details, click [here](#).

Azure SDK for .NET 2.9 release notes

9/19/2017 • 3 min to read • [Edit Online](#)

This topic includes release notes for versions 2.9 and 2.9.6 of Azure SDK for .NET.

Azure SDK for .NET 2.9.6 release summary

Release date: 11/16/2016

No breaking changes to the Azure SDK 2.9 have been introduced in this release. There is also no upgrade process needed to leverage this SDK with existing Cloud Service projects.

Visual Studio 2017 Release Candidate

- In Visual Studio 2017 RC, this release of the Azure SDK for .NET is built in in the Azure Workload. All the tools you need to do Azure development will be part of Visual Studio 2017 RC going forward. For Visual Studio 2015 and Visual Studio 2013, the SDK will still be available through WebPI. We will be discontinuing Azure SDK for .NET releases for Visual Studio 2013, when Visual Studio 2017 releases as a final product. Follow this link to download Visual Studio 2017 RC: <https://www.visualstudio.com/vs/visual-studio-2017-rc/>

Azure Diagnostics

- Changed the behavior to only store a partial connection string with the key replaced by a token for Cloud Services diagnostics storage connection string. The actual storage key is now stored in the user profile folder so its access can be controlled. Visual Studio will read the storage key from user profile folder for local debugging and publishing process.
- In response to the change described above, Visual Studio Online team enhanced the Azure Cloud Services deployment task template so users could specify the storage key for setting diagnostics extension when publishing to Azure in Continuous Integration and Deployment.
- We've made it possible to store secure connection string and tokenization for Azure Diagnostics (WAD), to help you solve problems with configuration across environments.

Windows Server 2016 virtual machines

- Visual Studio now supports deploying Cloud Services to OS Family 5 (Windows Server 2016) virtual machines. For existing cloud services, you can change your settings to target the new OS Family. When creating new cloud services, if you choose to create the service using .net 4.6 or higher, it will default the service to use OS Family 5. For more information, you can review the [Guest OS Family support table](#).

Known issues

- Azure .NET SDK 2.9.6 introduced a restriction that blocks deployment of projects using unsupported .NET frameworks (such as .NET 4.6) to any OS Family < 5. A workaround is provided [here](#).

Azure In-Role Cache

- Support for Azure In-Role Cache ends on November 30, 2016. For more details, click [here](#).

Azure Resource Manager Templates for Azure Stack

- We've introduced Azure Stack as a deployment target for your Azure Resource Manager Templates.

Azure SDK for .NET 2.9 summary

Overview

This document contains the release notes for the Azure SDK for .NET 2.9 release.

For detailed information about updates in this release, see the [Azure SDK 2.9 announcement post](#).

Azure SDK 2.9 for Visual Studio 2015 Update 2 and Visual Studio "15" Preview

This update includes the following bug fixes:

- Issue related to REST API Client Generation in which the string "Unknown Type" would appear as the name of the code-gen folder and/or the name of the namespace dropped into the generated code.
- Issue related to Scheduled WebJobs in which the authentication information was failing to be passed to the Scheduler provisioning process.

This update includes the following new feature:

- Support for secondary App Services in the "Services" tab of the App Service provisioning dialog.

Azure Data Lake Tools for Visual Studio 2015 Update 2

This update includes the following:

- **Azure Data Lake Tools** for Visual Studio is now merged into the Azure SDK for .NET release. The tool is automatically installed when you install Azure SDK.
The tool is updated frequently, go [here](#) to get the updates.
- **Server Explorer** now enables you to view all and create some U-SQL metadata entities. For more information, see [this](#) blog.

HDInsight Tools

HDInsight Tools for Visual Studio now supports HDInsight version 3.3, including showing Tez graphs and other language fixes.

Azure Resource Manager

This release adds [KeyVault](#) support for Resource Manager templates.

See also

[Azure SDK 2.9 announcement post](#)

Azure SDK for .NET 2.8, 2.8.1 and 2.8.2

9/19/2017 • 4 min to read • [Edit Online](#)

Overview

This article contains the release notes (that includes known issues and breaking changes) for the Azure SDK for .NET 2.8, 2.8.1 and 2.8.2 releases.

For complete list of new features and updates made in this release, see the [Azure SDK 2.8 for Visual Studio 2013 and Visual Studio 2015 announcement](#).

Azure SDK for .NET 2.8

Download Azure SDK for .NET 2.8

[Azure SDK for .NET 2.8 for Visual Studio 2015](#)

[Azure SDK for .NET 2.8 for Visual Studio 2013](#)

.NET 4.5.2 support

Known issues

Azure .NET SDK 2.8 allows you to create .NET 4.5.2 Cloud Service packages. However .NET 4.5.2 framework will not be installed on the default Guest OS images until January 2016 Guest OS release. Before that, .NET 4.5.2 framework will be available through a separate Guest OS release version – November 2015-02. See the [Azure Guest OS Releases and SDK Compatibility Matrix](#) page to track when the image will be released. Once the November 2015-02 image is released you can choose to use that image by updating your Cloud Service configuration file (.cscfg) file. In the service configuration file set the osVersion attribute of the ServiceConfiguration element to the string "WA-GUEST-OS-4.26_201511-02". If you choose to opt in to use this image then you will no longer get automatic updates to the Guest OS. To get the automatic updates the osVersion must be set to "*" and .NET 4.5.2 will only be available through automatic updates in January 2016.

Azure Data Factory

Known issues

During a **Data Factory Template** project creation involving sample data, azure power shell script may fail if azure power shell version installed on the machine is after 0.9.8.

In order to successfully create this type of project, you must install [azure power shell version 0.9.8](#).

Azure Resource Manager Tools

Breaking changes

The PowerShell script provided by the Azure Resource Group project has been updated in this release to work with the new Azure PowerShell cmdlets version 1.0. This new script will not work from with Visual Studio when using a version of the SDK prior to 2.8.

Scripts from projects created in earlier versions of the SDK will not run from within Visual Studio when using the 2.8 SDK. All scripts will continue to work outside of Visual Studio with the appropriate version of the Azure PowerShell cmdlets.

The 2.8 SDK requires version 1.0 of the Azure PowerShell cmdlets. All other versions of the SDK require version 0.9.8 of the Azure PowerShell cmdlets. For more information see [this blog](#).

Web Tools Extensions

Known issues

The following known issues will be addressed in the following release.

- App Service related Cloud and Server Explorer gesture for non-production environments (like Azure China or Azure Stack customers) do not work. For customers in these impacted areas, downloading the publish profile from the Azure portal will enable publishing ability. A future release will repair gestures such as "Attach Debugger" and "View Streaming Logs" for Azure China and Stack customers.
- Customers may see errors during App Service creation when the App Insights instance to which they are deploying is in a region other than East US. In these scenarios, creating an App Service in the portal and downloading the publish profile will enable publishing scenarios.

Azure HDInsight Tools

New updates

- You can execute your Hive query in the cluster via HiveServer2 with almost no overhead and see the job logs in real-time.
- Using the new Hive Task Execution View you can dig into your job deeper, find more details, and identify potential issues.

For information, see [Azure SDK 2.8 for Visual Studio 2013 and Visual Studio 2015](#).

Azure SDK for .NET 2.8.1

Known Issues for Visual Studio 2013 and Visual Studio 2015

1. Triggered WebJob publishes to slots will show an error and won't set a schedule, but it will push the WebJob to Azure. Customers who are in need of a Scheduled job can then use the Azure Portal to set up the schedule for the WebJob.
2. Python customers may experience debugger issues. Service team is rolling out a fix for this but if customers are affected, please let Microsoft know in the forums or on the announcement blog or release notes comments section.
3. Customers in certain regions (such as South India) will experience App Service provisioning errors. This is consistent with the portal, and customers who experience this issue can use the Azure portal to request access to publish to these geo-regions. Once they request access to these regions using the Azure portal provisioning should work.

Azure SDK for .NET 2.8.2

Following the installation of the 2.8.2 tools, customers may experience the following issue.

- If you are using Windows 10 and have not installed Internet Explorer, you may get an "Internet Explorer could not be found" error. To resolve the issue, install Internet Explorer using the Add/Remove Windows Components dialog.

If you observe this issue, use the Send-a-smile feature to report it.

For more information, see [this](#) post.

Other updates

For other updates, see [Azure SDK 2.8 announcement post](#).

Also see

[Azure SDK 2.8 announcement post](#)

[Support and Retirement Information for the Azure SDK for .NET and APIs](#)

Azure SDK for .NET 2.7 and .NET 2.7.1 Release Notes

9/19/2017 • 7 min to read • [Edit Online](#)

Overview

This document contains the release notes for the Azure SDK for .NET 2.7 release.

The document also contain the release notes for the Azure SDK for .NET 2.7.1 release.

Azure SDK 2.7 is only supported in Visual Studio 2015 and Visual Studio 2013. [Azure SDK 2.6](#) is the last supported SDK for Visual Studio 2012.

For detailed information about this release, see [Azure SDK 2.7 announcement post](#) and [Azure SDK 2.7.1 announcement post](#).

Azure SDK for .NET 2.7

Sign in improvements for Visual Studio 2015

Azure SDK 2.7 for Visual Studio 2015 supports the new identity management features in Visual Studio 2015. This includes support for accounts accessing Azure via Role Based Access Control, Cloud Solution Providers, DreamSpark and other account and subscription types.

The sign-in improvements included with Azure SDK 2.7 are only available in Visual Studio 2015. Support for Visual Studio 2013 is included in Azure SDK 2.7.1.

Mobile SDK

Updated **Mobile Apps** templates to reflect newest [NuGet package](#) and setup process.

Service Bus

General bug fixes and improvements. For detail on updates and features, please refer to the release notes of the latest [Service Bus NuGet](#).

HDIInsight Tools

In this release the following updates were made. These updates are in preview. For more information, see [this blog](#).

- Hive graphs for Hive on Tez jobs
- Full Hive DML IntelliSense support
- Pig template support
- Storm templates for Azure services

Breaking changes

- Old **Storm** project must be upgraded when using this version of the tools. For more information, see [this blog](#).
- Visual Studio Web Express is no longer supported. For more information, see [this blog](#).

Azure App Service Tools

In this release the following updates were made to Web Tools Extensions. For more information see [this blog](#).

- Support for DreamSpark accounts added
- Full change to Azure Tools made to support the new Azure Resource Management APIs
- Support for Azure App Service added to [Cloud Explorer](#)

Known issues

Web App deployment slot nodes don't appear under the Slots node in Server Explorer, and Web App deployment slot child nodes don't load under Cloud Explorer. This issue has been resolved and prepared for the next SDK release.

Cloud Explorer for Visual Studio 2015

Azure SDK 2.7 includes Cloud Explorer for Visual Studio 2015 which enables you to view your Azure resources, inspect their properties, and perform key developer actions from within Visual Studio.

Cloud explorer supports the following:

- Resource Group and Resource Type views of your Azure resources
- Search for resources by name (available in Resource Type view)
- Support for subscriptions and resources that have Role Based Access Control (RBAC) applied
- Integrated Action panel which shows developer-focused actions specific to selected resources. For example: Attach remote debugger for Virtual Machines created using the Resource Manager Stack, View diagnostics data for Virtual Machines etc.
- Integrated Properties panel which shows developer-focused properties commonly needed during dev/test
- Quick switching of the account to use when enumerating resources (use Settings command on toolbar)
- Filtering of subscriptions to use when enumerating resources (use Settings command on toolbar)
- Deep links to the Azure Portal for management of resources and resource groups

Azure Resource Manager Tools

The Azure Resource Manager Tools have been updated to work with Role Based Access Control (RBAC) and new subscription types. Included with these changes is the ability to use new storage accounts, in addition to classic storage, to store artifacts during deployment.

If you're using an Azure Resource Group project from a previous version of the SDK with the SDK 2.7, a new deployment script is needed to deploy using a new storage account instead of classic storage. You will be prompted before changes are made to your project to add the new script. The old script will be renamed and you will need to manually make any modifications to the new script.

Storage Explorer Tools

- Support for viewing Append Blobs. More info in [this blog post](#).
- Support for viewing Premium Storage accounts through Server Explorer. Server Explorer will only display page blobs for premium storage accounts as they are the only supported type for premium storage accounts.

Azure Data Factory Tools for Visual Studio

Introducing **Azure Data Factory Tools** for Visual Studio. Below are the enabled features. See [this blog](#) for more information.

- **Template based authoring:** Select use-cased based templates, data movement templates or data processing templates to deploy an end-to-end data integration solution and get started hands-on quickly with Data Factory.
- **Integration with Solution Explorer for authoring and deploying Data Factory entities:** Create & deploy pipelines and related entities as Visual Studio projects.
- **Integration with Diagram view for visual interaction while authoring:** Visually author pipelines and datasets with aid from the Diagram view.
- **Integration with Server explorer for browsing and interaction with already deployed entities:** Leverage the Server Explorer to browse already deployed data factories and corresponding entities. Import a deployed data factory or any entity (Pipeline, Linked Service, Datasets) into your project.
- **JSON editing with schema validation and rich intellisense:** Efficiently configure and edit JSON documents of Data Factory entities with rich intellisense and schema validation
- **Multi-Environment publishing:** Publish authored pipelines to dev, test or Prod environment by creating separate config files for each environment.

- **Pig, Hive and .Net based Data Processing Support:** Support for Pig and Hive Scripts in Data Factory project.
Support for referencing C# Project for managing .Net Activity.

Azure SDK for .NET 2.7.1

The following section contains updates that were introduced with the Azure SDK for .NET 2.7.1 release.

HDInsight Tools

For more detailed explanation about HDInsight tools updates, see [this blog](#).

- Hive Job Operator View (a new feature)

To help you understand your Hive query better, the Hive Operator View feature was added. To see all the operators inside a vertex, double click on the vertices of the job graph. To view more details of a particular operator, hover over that operator.

- Hive Error Marker (a new feature)

To enable you to view the grammar errors instantly, the Hive Error Marker feature was added. Also, error messages were enhanced and you can now see detailed grammar errors instantly (until this release, you had to submit a Hive script to the cluster and wait for some time before getting details about your error message).

- Storm Topology Graph (a new feature)

Visualizing is very important when you want to see if your topology is working as expected. In this release we added visualization for Storm graphs. You can visualize the important metrics for your topology (for example, a color indicates whether a certain Bolt is "busy" or not). You can also double click the Bolt/Spout to view more details.

- Support for HDInsight clusters that were created in the Azure Portal (a bug fix)

You can now use Visual Studio to view and submit jobs to all your HDInsight clusters no matter where the cluster were created.

- More IntelliSense Support& Faster Hive Metadata Loading (an improvement)

We have improved the IntelliSense by adding more user friendly suggestions. For example, table alias can now also be suggested in IntelliSense so you can write your query more easily. Also, we have improved the Hive metadata loading so it will just take several seconds to list all the databases, tables and columns of your Hive metastore.

For more detailed explanation about HDInsight tools updates, see [this blog](#).

Improvements in Visual Studio 2013

- Azure SDK 2.7.1 enables Visual Studio 2013 to access Azure accounts and subscriptions via Role Based Access Control, Cloud Solution Providers, and Dreamspark.
- With Azure SDK 2.7.1, the new Cloud Explorer tool window is now also available in Visual Studio 2013.

Known issues

Installing the Azure SDK 2.6 or 2.7.1 for Visual Studio Community 2013 on a non-English OS will display a warning that the English and non-English resources of Visual Studio may be mismatched. This warning may be safely dismissed. It will only occur if the machine did not contain a prior installation of Visual Studio Community 2013 and you are installing the SDK on a non-English OS. The warning is shown after the language pack applies the RTM resources to Visual Studio, but before it applies Update 4. Dismissing the warning will allow the language pack to continue and complete applying the Update 4 version of the language pack content.

LightSwitch projects are not compatible with this release. This issue will be resolved with the next SDK release.

Also see

[Azure SDK 2.7.1 announcement post](#)

[Azure SDK 2.7 announcement post](#)

[Support and Retirement Information for the Azure SDK for .NET and APIs](#)

Azure SDK for .NET 2.6 Release Notes

9/19/2017 • 4 min to read • [Edit Online](#)

This document contains the release notes for the Azure SDK for .NET 2.6 release.

With Azure SDK 2.6 you can develop cloud service applications (PaaS) targeting .NET 4.5.2 or .NET 4.6 provided that you manually install the target .NET Framework on the Cloud Service Role. See [Install .NET on a Cloud Service Role](#).

Service Bus updates

- Event Hubs:
 - Now allows targeted access control when sending events by exposing additional publisher endpoint for Event Hubs.
 - Additional stability and improvement added to Event Hubs feature.
 - Adding support of Amqp protocol over WebSocket for messaging and Event Hubs.

HDInsight Tools for Visual Studio updates

- **IntelliSense enhancement:** remote metadata suggestion

Now HDInsight Tools for Visual Studio supports getting remote metadata when you are editing your Hive script. For example, you can type **SELECT * FROM** and all the table names will be shown. Also, the column names will be shown after specifying a table.

- **HDInsight emulator support**

Now HDInsight Tools for Visual Studio support connecting to HDInsight emulator, so you could develop your Hive scripts locally without introducing any cost, then execute those scripts against your HDInsight clusters.

For more information, refer to [this manual](#).

- **HDInsight Tools for Visual Studio support for generic Hadoop clusters (Preview)**

Now HDInsight Tools for Visual Studio support generic Hadoop clusters, so you can use HDInsight Tools for Visual Studio to do the following:

- connect to your cluster,
- write Hive query with enhanced IntelliSense/auto-completion support,
- view all the jobs in your cluster with an intuitive UI.

For more information, refer to [this manual](#).

In-Role Cache updates

- **In-Role Cache** was updated to use **Microsoft Azure Storage SDK** version 4.3. Until now, the **In-Role Cache** was using Azure Storage SDK version 1.7.

Customers using Azure SDK 2.5 or below should update to Azure SDK 2.6 and move to the new version of Azure Storage SDK.

At this time Azure Storage version 2011-08-18 is scheduled to be removed August 1, 2016. Any migrations of In-Role Cache from Azure SDK 2.5 or below to 2.6 must be complete by this time. For more information

on the retirement of Azure Storage version 2011-08-18, see [Microsoft Azure Storage Service Version Removal Update: Extension to 2016](#).

IMPORTANT

We're announcing the November 30, 2016, retirement for Azure Managed Cache Service and Azure In-Role Cache. We recommend that you migrate to Azure Redis Cache in preparation for this retirement. For more information on dates and migration guidance, see [Which Azure Cache offering is right for me?](#)

Azure App Service Tools

The following items were updated in the Azure SDK 2.6 release.

- Azure publishing enhanced to include Azure API Apps as a deployment target.
- API Apps provisioning functionality to enable users with API App creation and provisioning functionality.
- Server Explorer changed to reflect new App Service node, with Web, Mobile, and API apps grouped by Resource Group.
- Add Azure API App Client gesture added to most C# projects that will enable automatic generation of Swagger-enabled API Apps running in a user's Azure subscription.
- API Apps tooling and App Service nodes in Server Explorer are available in Visual Studio 2013 only.

Azure Resource Manager Tools updates

The Azure resource manager tools have been updated to include templates for Virtual Machines, Networking and Storage. The JSON editing experience has been updated to include a new outline view for templates and the ability to edit the templates using JSON snippets. Templates deployed from Visual Studio use a PowerShell script provided with the project, so any changes made to the script will be used by Visual Studio.

Diagnostics improvements for Cloud Services

Azure SDK 2.6 brings back support for collecting diagnostics logs in the Azure compute emulator and transferring them to development storage. Any diagnostics logs (including application trace Logs, Event Tracing for Windows (ETW) logs, performance counters, infrastructure logs and windows event logs) generated when the application is running in the emulator can be transferred to development storage to verify that your diagnostics logging is working on your local machine.

The Diagnostics storage account can now be specified in the service configuration (.cscfg) file making it easier to use different diagnostics storage accounts for different environments. There are some notable differences between how the connection string worked in Azure SDK 2.4 and Azure SDK 2.6. For more information on how to use the Diagnostics Storage connection string and how it impacts your projects see [Configuring Diagnostics for Azure Cloud Services](#).

Breaking changes

Azure Resource Manager Tools

- The **Cloud Deployment Projects** project type available in the Azure SDK 2.5 has been renamed to **Azure Resource Group**.
- **Cloud Deployment Projects** type of projects created in the Azure SDK 2.5 can be used in 2.6 but deploying the template from Visual Studio will fail. However, deploying with the PowerShell script will still work as it did previously. For information on how to use **Cloud Deployment Projects** in 2.6 read this [post](#).

Known issues

- Collecting diagnostics logs in the emulator requires a 64-bit operating system. Diagnostics logs will not be collected when running on a 32-bit operating system. This does not affect any other emulator functionality.
- Azure SDK 2.6 released on 4/29/2015 had two issues:
 - Universal App could not be loaded in Visual Studio 2015 when Azure SDK 2.6 was installed on the machine.
 - Debugging a Cloud Service project would fail in Visual Studio 2013 and Visual Studio 2015 where Visual Studio becomes unresponsive and crashes while displaying a dialog box with the message "Configuring diagnostics for emulator".

An update to Azure SDK 2.6 was released on 5/18/2015. The updated version is 2.6.30508.1601; it contains fixes for two issues described above. You can identify the build of the SDK from Control Panel -> Programs and Features -> Microsoft Azure Tools for Microsoft Visual Studio 2013 – v 2.6. The Version column will display the build number.

If you are still facing the above issues, install the latest version of the Azure 2.6 SDK for [VS 2012](#), [VS 2013](#) or [VS 2015](#).

See Also

[Support and Retirement Information for the Azure SDK for .NET and APIs](#)

Azure SDK for .NET 2.5.1 Release Notes

11/2/2017 • 3 min to read • [Edit Online](#)

This document contains the release notes for the Azure SDK for .NET 2.5.1 release.

Azure SDK for .NET 2.5.1 release notes

The following are new features and updates in the Azure SDK for .NET 2.5.1.

- New features\scenarios related to **Web Tools Extensions**.
 - Azure Websites was renamed to Azure App Service.
 - Azure API Apps (Preview) support has been added so that customers can publish ASP.NET projects as API Apps, and then use the Add > Azure API App Client gesture in C# projects to generate code based on the structure of the deployed API App.
 - The Websites node in Server Explorer has been deprecated in lieu of the Azure App Service node, which contains support for Resource Group-based grouping of Azure API Apps, Mobile Apps, and Web Apps.
 - Azure Mobile Apps (Preview) support has been added so that customers can create new Mobile Apps projects, add Mobile Apps controllers, publish the projects, and remotely debug applications.
 - Add > Azure API App Client gesture now supports local Swagger JSON files, so Web API developers can use third-party NuGets like Swashbuckle to generate Swagger or author it manually. This way, client developers can use the code-generation features to consume any Swagger endpoint in C# projects.
 - Web App and API App publishing dialogs have been enhanced to support the Azure portal concept of resource grouping, and selection/creation of Azure Resource Groups and App Service Plans are represented in the new Web App and API App provisioning dialog.
 - Azure API App Server Explorer nodes provide links to the API Apps deep link in the Azure portal, as well as other features like Log Streaming and Remote Debugging.

For known issues and current limitations in Azure SDK .NET 2.5.1 [this](#) section below.

- New features\scenarios related to **HDInsight Tools** in Visual Studio are enabled in this release.
 - Local validation of hive scripts. Click the Validate script button in the toolbar to see if there are any errors in your script.
 - Improved debugging of Hive jobs. You can now debug Hive jobs by accessing Yarn logs in Visual Studio. If your application has performance issues, investigating YARN logs provide useful information..
 - (Public Preview) Keyword auto-completion and IntelliSense support for Hive. To help you author Hive scripts, HDInsight Tools for Visual Studio added keyword auto-completion and IntelliSense support for Hive.
 - Storm support. You can now use HDInsight Tools for Visual Studio to develop Storm topologies/Spouts/Bolts in C#. You can then submit the developed topology to a Storm cluster and see the topology/bolt/spout status. You can use system logs and customer logs to troubleshoot your Storm topologies/Bolts/Spouts. You can also use existing JAVA assets in Storm on HDInsight.

For more information, see [Get started using HDInsight Hadoop Tools for Visual Studio](#).

Azure SDK for .NET 2.5.1 known issues and limitations

- Azure API Apps is visible as a deployment target for Mobile Apps. Web Apps should be the only destination for Mobile Apps until a subsequent release.

- Azure API App provisioning can result in success but intermittently fail to update the progress in the Azure App Service Activity window. Workaround is to check status of the new Azure API App in the Azure portal.
- File > New Project > API App > F5 experience results in an HTTP error because there is no default/index.html. Workaround is to manually browse to the /api/values URL.
- Intermittently, Server Explorer icons appear flattened. Restarting VS resolves this issue.
- If an exception is thrown during Web App or API App provisioning (such as exceeded quota errors or duplicate Azure API App gateway name), the errors show some raw JSON text.
- Intermittent project-creation issues when Application Insights is checked at project creation time.
- Occasionally, the generated Azure API App Client code is missing namespaces, they need to be manually included (or automatically imported via Visual Studio cues) for code to compile.
- Mobile App projects should be published to web apps, but you must pick a site you created as a Mobile App in the Azure portal since Mobile App projects require a database.
- The start page for Mobile Apps uses the term "mobile service" instead of "mobile apps"
- Mobile App project creation may take up to a minute to create.
- During API App provisioning (in some cases) an error comes back from the Azure API reflecting that the permissions could not be set properly, while the API App has been properly provisioned and is ready for use. You can manually set permissions using the Azure portal.
- Application Insights is not supported on API App templates and Mobile App templates.
- API App projects cannot be used in conjunction with Cloud Service projects.
- API App project templates are only available in C#.
- API App consumption via the "Add Azure API App Client" context menu is only supported in C#.

Best Practices for Azure App Service

11/13/2017 • 4 min 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 the effects can include the following:

- 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 you should 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 leveraging 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 statefull, scaling up is the only option, while if your application is stateless, scaling out will give you more flexibility and higher scale potential.

For more information about “statefull” vs “stateless” applications you can watch this video: [Planning a Scalable End-to-End Multi-Tier Application on Microsoft Azure Web App](#). For more information about App Service scaling and autoscaling options read: [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 in the case of a higher level protocol such as HTTP - Keep-Alive not being leveraged. Please 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 Nodejs and many outgoing http requests, dealing with HTTP - Keep-Alive is really important.

You can use the [agentkeepalive](#)  package to make it easier in your code.

You should always handle the `http` response, even if you do nothing in the handler. If you don't handle the response properly, your application will eventually get stuck because no more sockets are available.

For example, when working with the `http` or `https` package:

```
var 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 this 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 (e.g. 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, please review most recent results to understand which type of failure is happening. In the case of storage access failures, please review and update the storage settings used in the backup configuration. In the case of database access failures, please 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 backup please see the [Back up a web app in Azure App Service](#) documentation.

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, you could review our best practices and troubleshooting steps. This documentation 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: [Best practices and troubleshooting guide for Node applications on Azure App Service](#).

4 min to read •

Troubleshoot a web app in Azure App Service using Visual Studio

11/22/2017 • 30 min to read • [Edit Online](#)

Overview

This tutorial shows how to use Visual Studio tools to help debug a web app in [App Service](#), by running in [debug mode](#) remotely or by viewing application logs and web server logs.

NOTE

Although this article refers to web apps, it also applies to API apps and mobile apps.

You'll learn:

- Which Azure web app management functions are available in Visual Studio.
- How to use Visual Studio remote view to make quick changes in a remote web app.
- How to run debug mode remotely while a project is running in Azure, both for a web 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 Azure web app that you set up in [Get started with Azure and ASP.NET](#). 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 2017.

The streaming logs feature only works for applications that target .NET Framework 4 or later.

Web app configuration and management

Visual Studio provides access to a subset of the web 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

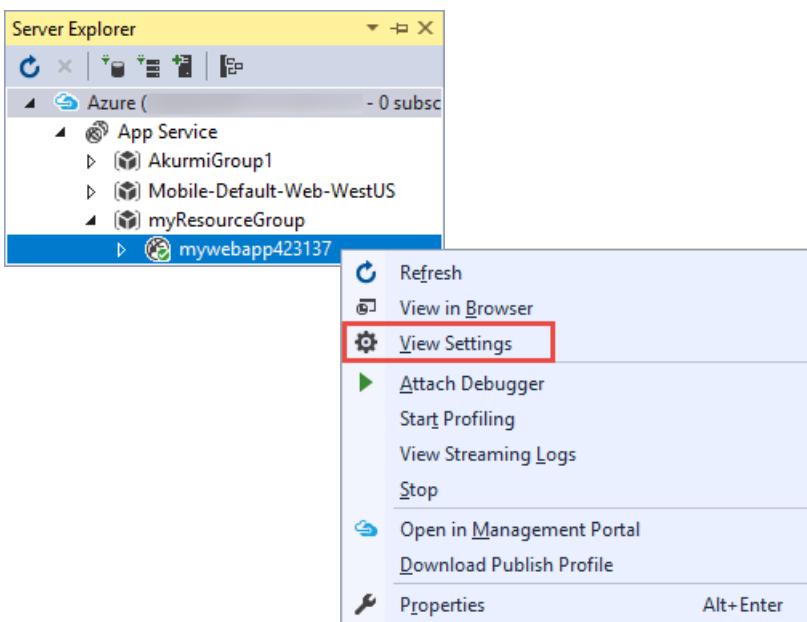
Certificates tab, and then click **Import**. Follow the directions to download and then import a subscription file (also called a *.publishsettings* file) for your Azure account.

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 web app that you created in [Create an ASP.NET web app in Azure][app-service-web-get-started-dotnet.md], and then right-click the web app node and click **View Settings**.



The **Azure Web App** tab appears, and you can see there the web app management and configuration tasks that are available in Visual Studio.

mywebapp423137: Azure Web App

Configuration Save Refresh

Actions

- Open in Management Portal
- Stop Web App
- Restart Web App

Web App Settings [Learn more](#)

.NET Framework Version	v4.5
Web Server Logging	Off
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Off
Remote Debugging	Off

Connection Strings

Name	Connection String	Database Type

Add Remove

Application Settings

Name	Value
WEBSITE_NODE_ID	6.9.1

Add Remove

In this tutorial, you'll use the logging and tracing drop-downs. You'll also use remote debugging but you'll use a different method to enable it.

For information about the App Settings and Connection Strings boxes in this window, see [Azure Web Apps: How Application Strings and Connection Strings Work](#).

If you want to perform a web 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 web 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

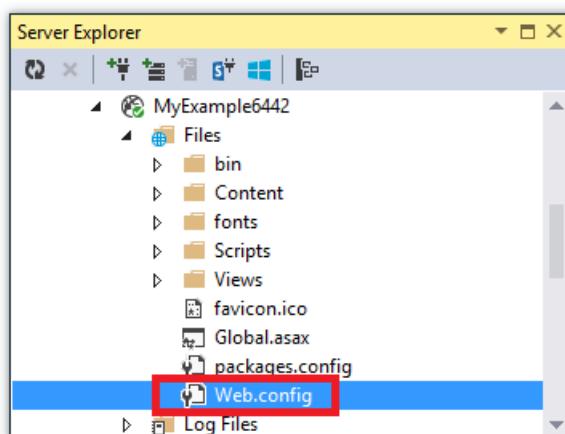
The screenshot shows a browser window with the URL <http://myexample6442.azurewebsites.net/>. The title bar says "HTTP 500 Internal Server Err...". The main content area has a blue header with an info icon and the text "The website cannot display the page". Below this, it says "Most likely causes:" with two bullet points: "The website is under maintenance." and "The website has a programming error.". It then asks "What you can try:" with three items: "Retype the address.", "Go back to the previous page.", and "More information". At the bottom, it states "This error (HTTP 500 Internal Server Error) means that the website you are visiting had a server problem which prevented the webpage from displaying." and "For more information about HTTP errors, see Help."

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 web app by using the *remote view* feature.

1. In **Server Explorer**, expand **Azure**, expand **App Service**, expand the resource group that your web app is located in, and then expand the node for your web app.

You see nodes that give you access to the web app's content files and log files.

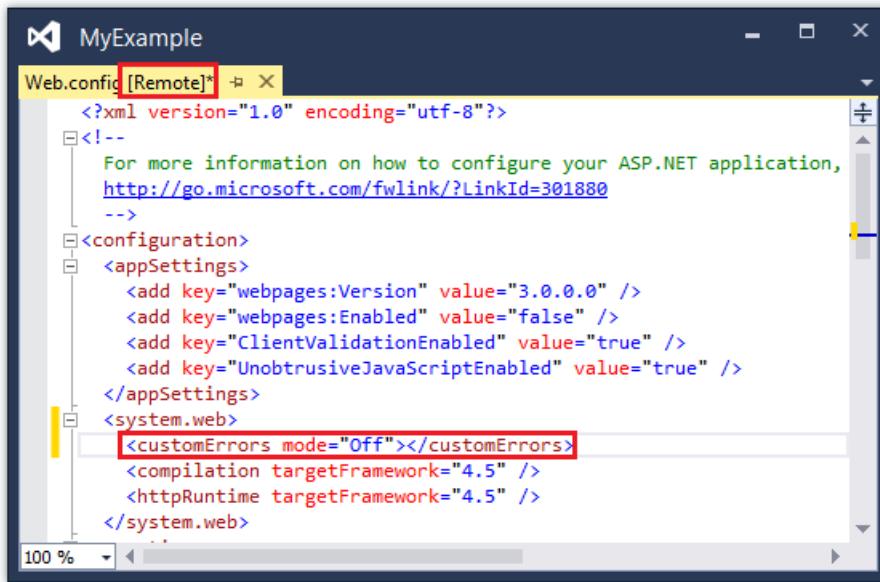
2. Expand the **Files** node, and double-click the *Web.config* file.



Visual Studio opens the *Web.config* file from the remote web 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:

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 Azure web app make troubleshooting easier.

Remote debugging web 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 web app in Azure][app-service-web-get-started-dotnet.md].

1. Open the web project that you created in [Create an ASP.NET web app in Azure][app-service-web-get-started-dotnet.md].

2. Open *Controllers\HomeController.cs*.

3. Delete the `About()` method and insert the following code in its place.

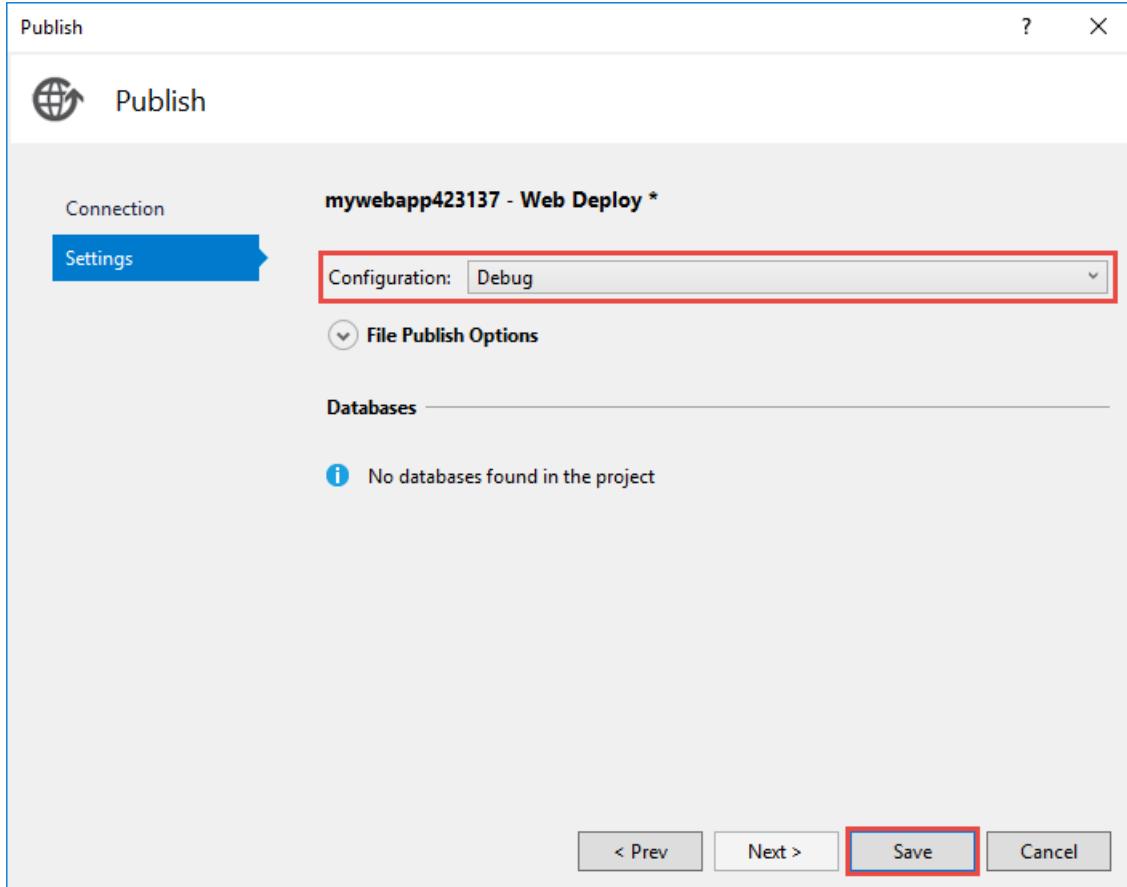
```
public ActionResult About()
{
    string currentTime = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
    ViewBag.Message = "The current time is " + currentTime;
    return View();
}
```

4. Set a breakpoint on the `ViewBag.Message` line.

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

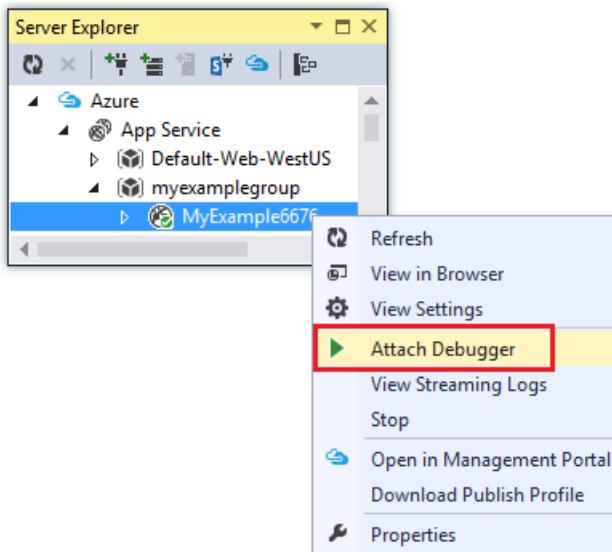
6. In the **Profile** drop-down list, select the same profile that you used in [Create an ASP.NET web app in Azure][app-service-web-get-started-dotnet.md]. Then, click Settings.

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 web app, close the browser.

9. In **Server Explorer**, right-click your web 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 a web 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.

```
public ActionResult About()
{
    string currentTime = DateTime.Now.ToString("T");
    ViewBag.Message = "The current time is " + currentTime;

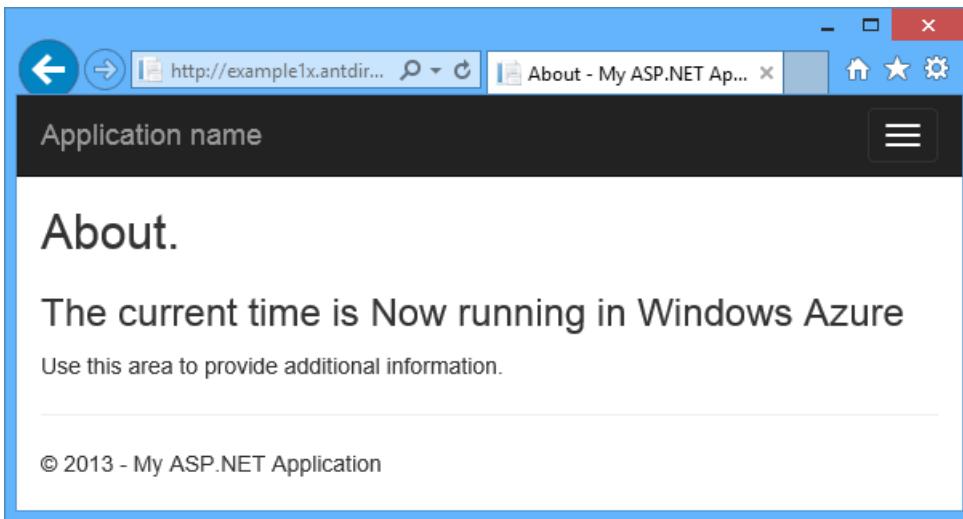
    return View();
}
```

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 web 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.

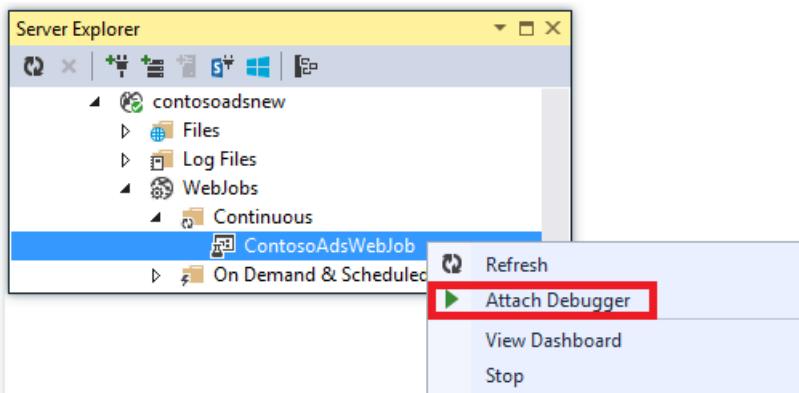
1. Open the web project that you created in [Get Started with the Azure WebJobs SDK](#).
2. In the ContosoAdsWebJob project, open *Functions.cs*.
3. Set a breakpoint on the first statement in the `GenerateThumbnail` method.

```
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 web app.

7. In **Server Explorer**, expand **Azure > App Service > your resource group > your web app > WebJobs > Continuous**, and then right-click **ContosoAdsWebJob**.
8. Click **Attach Debugger**.

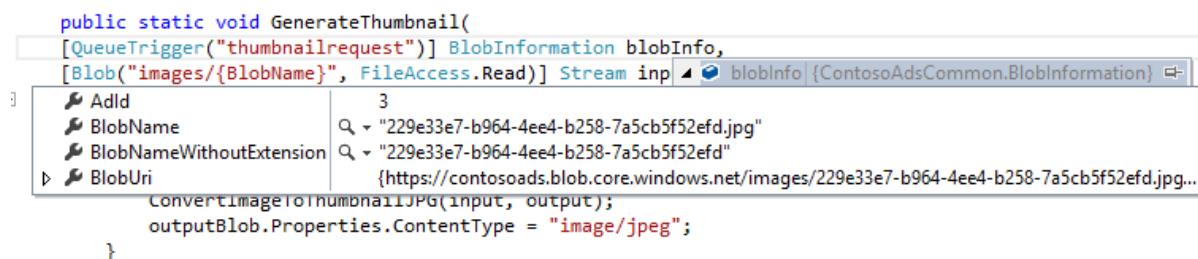


The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on a web app in a 48-hour period. When you start debugging again in the same period, there isn't a delay.

9. In the web browser that is opened to the Contoso Ads home page, create a new ad.

Creating an ad causes a queue message to be created, which is picked up by the WebJob and processed. When the WebJobs SDK calls the function to process the queue message, the code hits your breakpoint.

10. When the debugger breaks at your breakpoint, you can examine and change variable values while the program is running in the cloud. In the following illustration, the debugger shows the contents of the blobInfo object that was passed to the `GenerateThumbnail` method.



11. Press F5 to continue running.

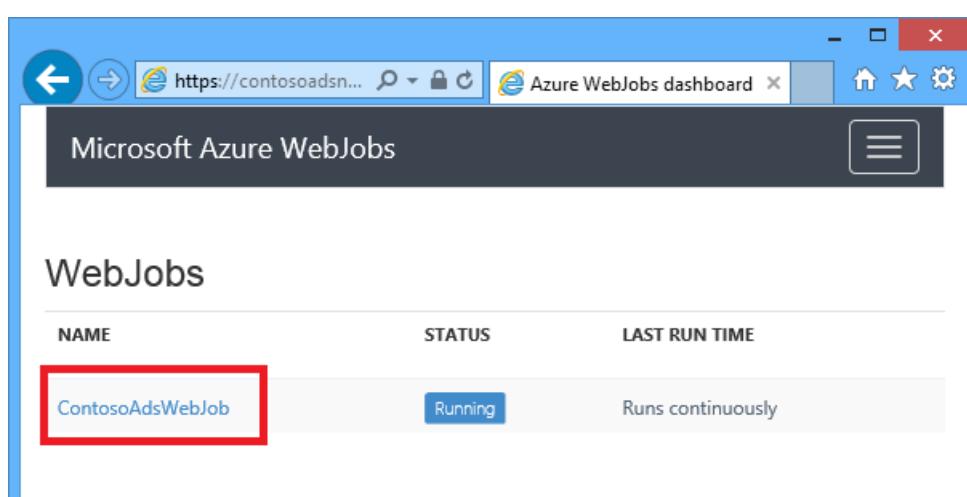
The `GenerateThumbnail` method finishes creating the thumbnail.

12. In the browser, refresh the Index page and you see the thumbnail.

13. In Visual Studio, press SHIFT+F5 to stop debugging.

14. In **Server Explorer**, right-click the ContosoAdsWebJob node and click **View Dashboard**.

15. Sign in with your Azure credentials, and then click the WebJob name to go to the page for your WebJob.



The Dashboard shows that the `GenerateThumbnail` function executed recently.

(The next time you click **View Dashboard**, you don't have to sign in, and the browser goes directly to the page for your WebJob.)

16. Click the function name to see details about the function execution.

The screenshot shows the Azure WebJobs dashboard with the URL <https://contosoadsns...>. 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". A green box indicates "Success 9 minutes ago (54 seconds running time)". Below this, 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	

At the bottom left is a button labeled "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 web 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 [Restrict stepping to Just My Code](#).
- 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 web 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 Azure web 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 web 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 web 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 web app, by default all logging is disabled.

Logs are written to files in a `LogFiles` folder in the file system of your web app and are accessible via FTP. Web server logs and application logs can also be written to an Azure Storage account. You can retain a greater volume of logs in a storage account than is possible in the file system. You're limited to a maximum of 100 megabytes of logs when you use the file system. (File system logs are only for short-term retention. Azure deletes old log files to make room for new ones after the limit is reached.)

Create and view application trace logs

In this section, you do the following tasks:

- Add tracing statements to the web project that you created in [Get started with Azure and ASP.NET](#).
- View the logs when you run the project locally.
- View the logs as they are generated by the application running in Azure.

For information about how to create application logs in WebJobs, see [How to work with Azure queue storage using the WebJobs SDK - How to write logs](#). The following instructions for viewing logs and controlling how they're stored in Azure apply also to application logs created by WebJobs.

Add tracing statements to the application

1. Open `Controllers\HomeController.cs`, and replace the `Index`, `About`, and `Contact` methods with the following code in order to add `Trace` statements and a `using` statement for `System.Diagnostics`:

```
public ActionResult Index()
{
    Trace.WriteLine("Entering Index method");
    ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";
    Trace.TraceInformation("Displaying the Index page at " + DateTime.Now.ToString());
    Trace.WriteLine("Leaving Index method");
    return View();
}

public ActionResult About()
{
    Trace.WriteLine("Entering About method");
    ViewBag.Message = "Your app description page.";
    Trace.TraceWarning("Transient error on the About page at " + DateTime.Now.ToString());
    Trace.WriteLine("Leaving About method");
    return View();
}

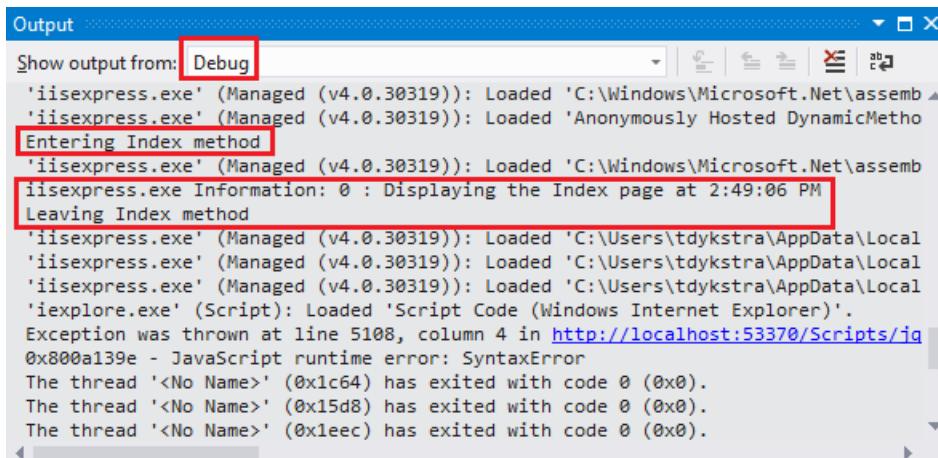
public ActionResult Contact()
{
    Trace.WriteLine("Entering Contact method");
    ViewBag.Message = "Your contact page.";
    Trace.TraceError("Fatal error on the Contact page at " + DateTime.Now.ToString());
    Trace.WriteLine("Leaving Contact method");
    return View();
}
```

2. Add a `using System.Diagnostics;` statement to the top of the file.

View the tracing output locally

1. Press F5 to run the application in debug mode.

The default trace listener writes all trace output to the **Output** window, along with other Debug output. The following illustration shows the output from the trace statements that you added to the `Index` method.



The screenshot shows the Visual Studio Output window with the title bar 'Output'. The dropdown menu 'Show output from' is set to 'Debug'. The window displays several lines of text, with specific lines highlighted by red boxes:

- 'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Windows\Microsoft.Net\assemb...
- 'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'Anonymously Hosted DynamicMetho...
- Entering Index method**
- 'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Windows\Microsoft.Net\assemb...
- iisexpress.exe Information: 0 : Displaying the Index page at 2:49:06 PM**
- Leaving Index method**
- 'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local...
- 'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local...
- 'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local...
- 'iexplore.exe' (Script): Loaded 'Script Code (Windows Internet Explorer)'.
- Exception was thrown at line 5108, column 4 in <http://localhost:53370/Scripts/jq...>
- 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`.

3. 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" />
```

4. Press CTRL+F5 to run the application.
5. 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>).
6. On the **Application Trace** page, click **View Details** on the first line (not the BrowserLink line).

Requests to this Application					Remaining: 9
No.	Time of Request	File	Status Code	Verb	
1	7/8/2013 10:41:38 AM		200	GET	View Details

The **Request Details** page appears, and in the **Trace Information** section you see the output from the trace statements that you added to the `Index` method.

The screenshot shows a browser window with the URL `localhost:53370/Trace.axd?id=0`. The main content area is titled "Request Details". It displays the following information:

Session Id:	7/8/2013 10:41:38 AM	Request Type:	GET
Time of Request:	AM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

Below this is a section titled "Trace Information". It contains a table with the following data:

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 web 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" />
```

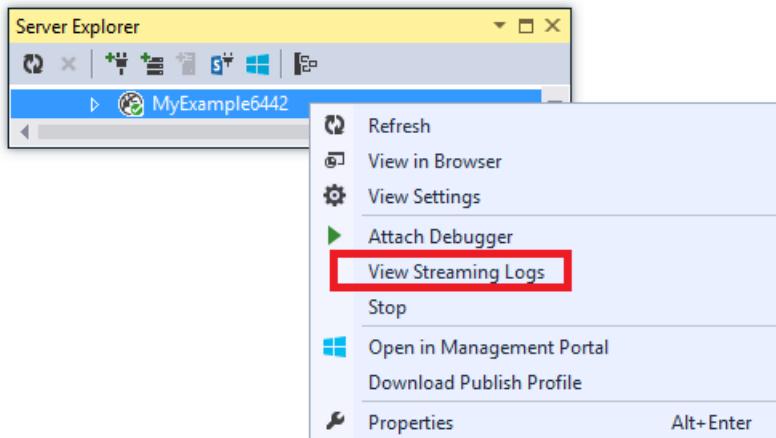
However, enabling `trace.axd` in a production web app is not recommended for security reasons. In the following sections, you'll see an easier way to read tracing logs in an Azure web app.

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 web 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.

```
Output
Show output from: Windows Azure Logs - example1z
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).

Web Publish Activity | Output
```

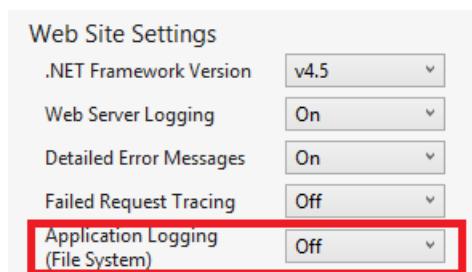
4. In the browser window that shows your application home page, click **Contact**.

Within a few seconds, the output from the error-level trace you added to the `Contact` method appears in the **Output** window.

```
Output
Show output from: Windows Azure Logs - example1z
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

Web Publish Activity | Output
```

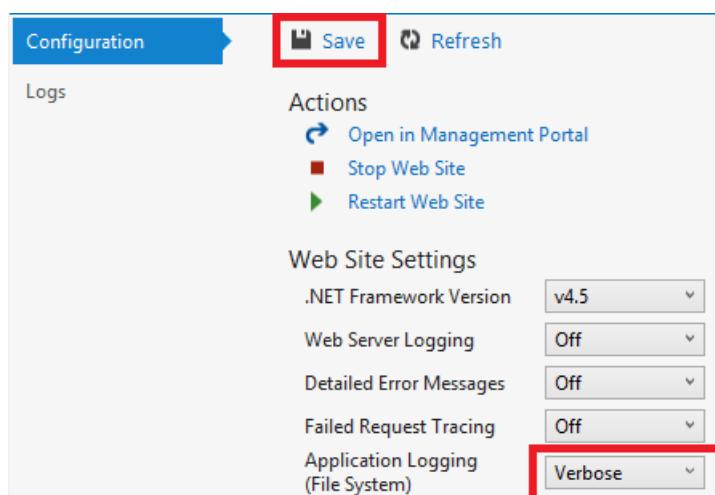
Visual Studio is only showing error-level traces because that is the default setting when you enable the log monitoring service. When you create a new Azure web app, all logging is disabled by default, as you saw when you opened the settings page earlier:



However, when you selected **View Streaming Logs**, Visual Studio automatically changed **Application Logging(File System)** to **Error**, which means error-level logs get reported. In order to see all of your tracing logs, you can change this setting to **Verbose**. When you select a severity level lower than error, all logs for higher severity levels are also reported. So when you select verbose, you also see information, warning, and error logs.

5. In **Server Explorer**, right-click the web app, and then click **View Settings** as you did earlier.

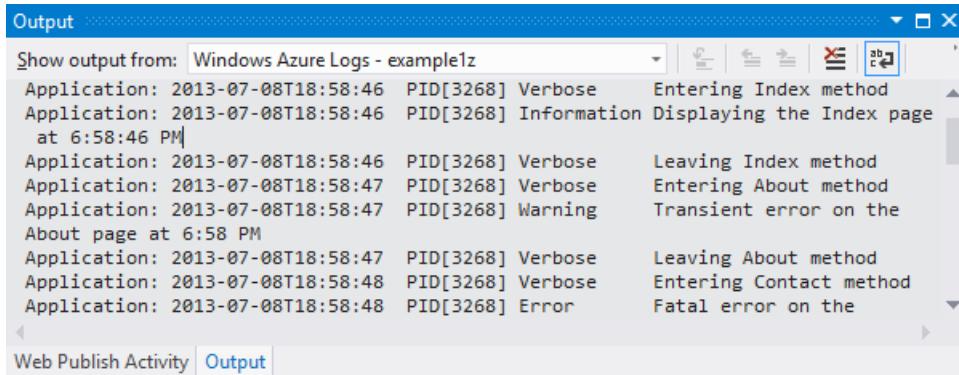
6. Change **Application Logging (File System)** to **Verbose**, and then click **Save**.



7. In the browser window that is now showing your **Contact** page, click **Home**, then click **About**, and then

click **Contact**.

Within a few seconds, the **Output** window shows all of your tracing output.



The screenshot shows the Microsoft Azure Logs tab of the Output window. The window title is "Output". The "Show output from:" dropdown is set to "Windows Azure Logs - example1z". The log entries are as follows:

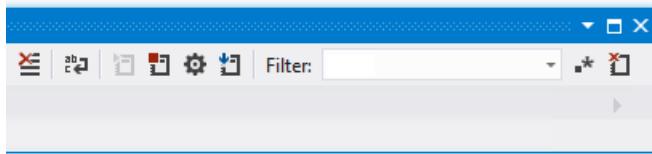
```
Application: 2013-07-08T18:58:46 PID[3268] Verbose Entering Index method
Application: 2013-07-08T18:58:46 PID[3268] Information Displaying the Index page
at 6:58:46 PM
Application: 2013-07-08T18:58:46 PID[3268] Verbose Leaving Index method
Application: 2013-07-08T18:58:47 PID[3268] Verbose Entering About method
Application: 2013-07-08T18:58:47 PID[3268] Warning Transient error on the
About page at 6:58 PM
Application: 2013-07-08T18:58:47 PID[3268] Verbose Leaving About method
Application: 2013-07-08T18:58:48 PID[3268] Verbose Entering Contact method
Application: 2013-07-08T18:58:48 PID[3268] Error Fatal error on the
```

At the bottom of the window, there are two tabs: "Web Publish Activity" and "Output". The "Output" tab is selected.

In this section, you enabled and disabled logging by using Azure web 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 web 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 web app. In order to see them in the **Output** window, you must enable them for the web 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

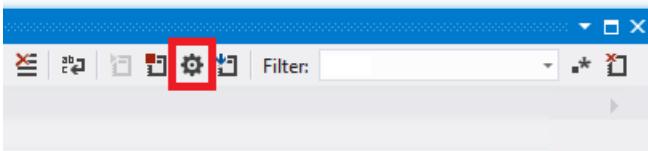
Logs Actions

- Open in Management Portal
- Stop Web Site
- Restart Web Site

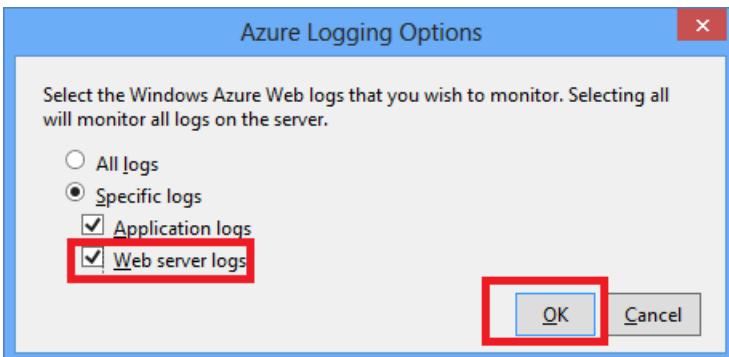
Web Site Settings

.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 web app, click **Home**, then click **About**, and then click **Contact**.

The application logs generally appear first, followed by the web server logs. You might have to wait a while for the logs to appear.

Output

Show output from: Windows Azure Logs - example1z

```

Connecting to Application logs ...
2013-07-08T19:50:34 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:51:34 No new trace in the past 1 min(s).
Application: 2013-07-08T19:52:34 No new trace in the past 2 min(s).
Connecting to Web server logs ...
2013-07-08T19:52:36 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:53:34 No new trace in the past 3 min(s).
Web server: 2013-07-08T19:53:36 No new trace in the past 1 min(s).
Application: 2013-07-08T19:54:34 No new trace in the past 4 min(s).
Web server: 2013-07-08T19:54:36 No new trace in the past 2 min(s).
Application: 2013-07-08T19:55:06 PID[3268] Information DotNetOpenAuth.Core,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=2780ccd10d57b246 (official)
Application: 2013-07-08T19:55:06 PID[3268] Information Reporting will use
isolated storage with scope: Domain, Assembly, Machine
Application: 2013-07-08T19:55:10 PID[3268] Verbose Entering Contact method
Application: 2013-07-08T19:55:10 PID[3268] Verbose Entering Index method
Application: 2013-07-08T19:55:10 PID[3268] Verbose Entering About method
Application: 2013-07-08T19:55:10 PID[3268] Warning Transient error on the
About page at 7:55 PM
Application: 2013-07-08T19:55:10 PID[3268] Verbose Leaving About method
Application: 2013-07-08T19:55:10 PID[3268] Error Fatal error on the
Contact page at 7:55:10 PM
Application: 2013-07-08T19:55:10 PID[3268] Verbose Leaving Contact method
Application: 2013-07-08T19:55:10 PID[3268] Information Displaying the Index page
at 7:55:10 PM
Application: 2013-07-08T19:55:10 PID[3268] Verbose Leaving Index method
Web server: 2013-07-08T19:55:36 No new trace in the past 3 min(s).
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET /Home/Contact X-ARR-LOG-
ID=1f7e79df-bfbcc-40c6-9573-cfcbb04612f93 80 - 131.107.0.112 Mozilla/5.0
+(compatible;+MSIE+10.0;+Windows+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 3307
623 7549
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET /Home/About X-ARR-LOG-
ID=8a35806d-8e92-479e-bad7-55d001c2fab2 80 - 131.107.0.112 Mozilla/5.0
+(compatible;+MSIE+10.0;+Windows+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 2845
619 8480
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET / X-ARR-LOG-ID=1e9c3f42-7f32-4a23-
a15b-037f2b69f870 80 - 131.107.0.112 Mozilla/5.0+(compatible;+MSIE+10.0;+Windows
+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 4137
599 9261
Application: 2013-07-08T19:56:34 No new trace in the past 1 min(s).
Application: 2013-07-08T19:57:34 No new trace in the past 2 min(s).
Web server: 2013-07-08T19:57:36 No new trace in the past 1 min(s).

```

Web Publish Activity | Output

By default, when you first enable web server logs by using Visual Studio, Azure writes the logs to the file system. As an alternative, you can use the Azure portal to specify that web server logs should be written to a blob container in a storage account.

If you use the portal to enable web server logging to an Azure storage account, and then disable logging in Visual Studio, when you re-enable logging in Visual Studio your storage account settings are restored.

View detailed error message logs

Detailed error logs provide some additional information about HTTP requests that result in error response codes (400 or above). In order to see them in the **Output** window, you have to enable them for the web app and tell Visual Studio that you want to monitor them.

1. In the **Azure Web App Configuration** tab that you opened from **Server Explorer**, change **Detailed Error Messages** to **On**, and then click **Save**.

Configuration

Save Refresh

Logs Actions

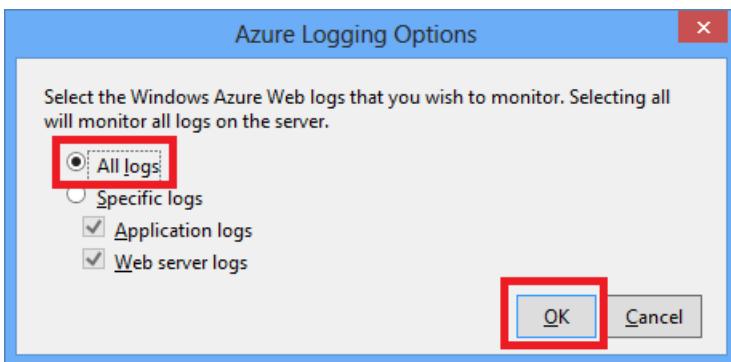
- Open in Management Portal
- Stop Web Site
- Restart Web Site

Web Site Settings

.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	On
Failed Request Tracing	Off
Application Logging (File System)	Verbose

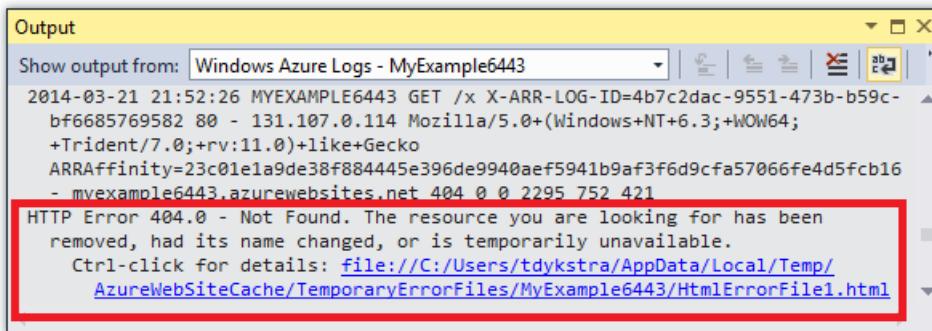
2. In the **Output** Window, click the **Specify which Microsoft Azure logs to monitor** button.

3. In the **Microsoft Azure Logging Options** dialog box, click **All logs**, and then click **OK**.



4. In the address bar of the browser window, add an extra character to the URL to cause a 404 error (for example, `http://localhost:53370/Home/Contactx`), and press Enter.

After several seconds, the detailed error log appears in the Visual Studio **Output** window.



Control+click the link to see the log output formatted in a browser:

The screenshot shows an IIS Detailed Error - 404.0 Not Found page. The title bar says "IIS Detailed Error - 404....". The main content area has several sections:

- HTTP Error 404.0 - Not Found**
- The resource you are looking for has been removed, had its name changed, or is temporarily unavailable.**
- Most likely causes:**
 - The directory or file specified does not exist on the Web server.
 - The URL contains a typographical error.
 - A custom filter or module, such as URLScan, restricts access to the file.
- Things you can try:**
 - Create the content on the Web server.
 - Review the browser URL.
 - Create a tracing rule to track failed requests for this HTTP status code and see which module is calling SetStatus. For more information about creating a tracing rule for failed requests, click [here](#).
- Detailed Error Information:**

Module	ManagedPipelineHandler	Request	http://example1z.azurewebsites.net:80/Home/Contactx
Notification	ExecuteRequestHandler	Physical Path	C:\DWASFiles\Sites\example1z\VirtualDirectory0\site\wwwroot\Home\Contactx
Handler	System.Web.Mvc.MvcHandler	Logon Method	Anonymous
Error Code	0x00000000	Logon User	Anonymous
- More Information:**

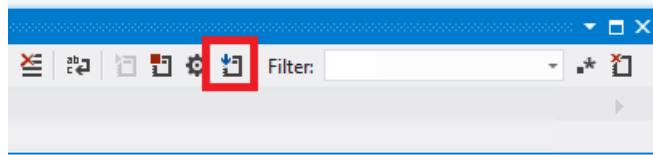
This error means that the file or directory does not exist on the server. Create the file or directory and try the request again.
[View more information >](#)

Microsoft Knowledge Base Articles:

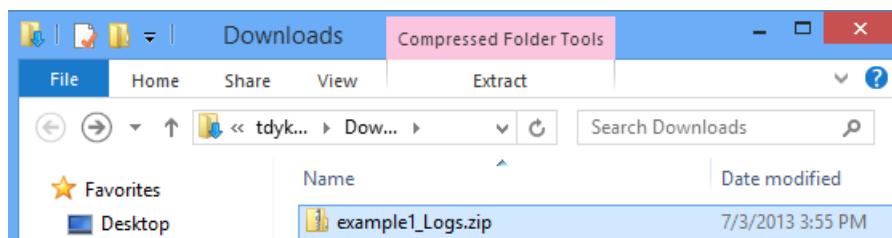
Download file system logs

Any logs that you can monitor in the **Output** window can also be downloaded as a .zip file.

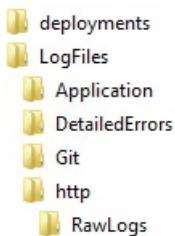
1. In the **Output** window, click **Download Streaming Logs**.



File Explorer opens to your *Downloads* folder with the downloaded file selected.



2. Extract the .zip file, and you see the following folder structure:



- Application tracing logs are in .txt files in the *LogFiles\Application* folder.
- Web server logs are in .log files in the *LogFiles\http\RawLogs* folder. You can use a tool such as [Log Parser](#) to view and manipulate these files.
- Detailed error message logs are in .html files in the *LogFiles\DetailedErrors* folder.

(The *deployments* folder is for files created by source control publishing; it doesn't have anything related to Visual Studio publishing. The *Git* folder is for traces related to source control publishing and the log file streaming service.)

View failed request tracing logs

Failed request tracing logs are useful when you need to understand the details of how IIS is handling an HTTP request, in scenarios such as URL rewriting or authentication problems.

Azure web apps use the same failed request tracing functionality that has been available with IIS 7.0 and later. You don't have access to the IIS settings that configure which errors get logged, however. When you enable failed request tracing, all errors are captured.

You can enable failed request tracing by using Visual Studio, but you can't view them in Visual Studio. These logs are XML files. The streaming log service only monitors files that are deemed readable in plain text mode: .txt, .html, and .log files.

You can view failed request tracing logs in a browser directly via FTP or locally after using an FTP tool to download them to your local computer. In this section, you'll view them in a browser directly.

1. In the **Configuration** tab of the **Azure Web App** window that you opened from **Server Explorer**, change **Failed Request Tracing** to **On**, and then click **Save**.

The screenshot shows the 'Configuration' tab of the Azure Web App window. At the top right are 'Save' and 'Refresh' buttons. Below is a 'Actions' section with 'Open in Management Portal', 'Stop Web Site', and 'Restart Web Site' options. The main area is 'Web Site Settings' with the following configuration:

.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	On
Failed Request Tracing	On
Application Logging (File System)	Verbose

2. In the address bar of the browser window that shows the web 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 web 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

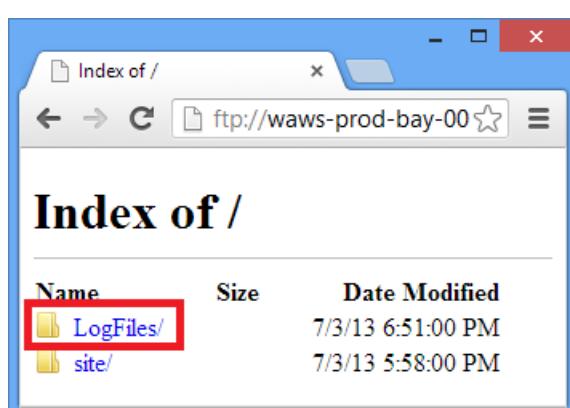
NOTE

When you log in, you have to use the full user name with the web 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".

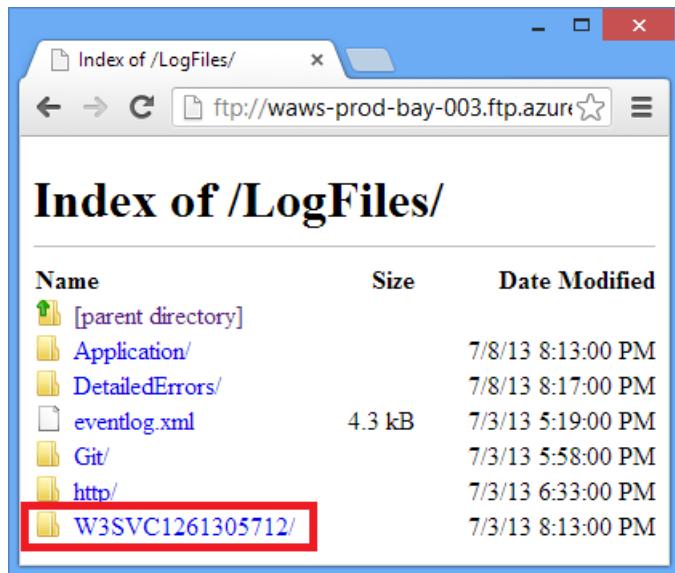
5. In a new browser window, go to the URL that is shown under **FTP hostname** or **FTPS hostname** in the **Overview** page for your web app.
6. Log in using the FTP credentials that you created earlier (including the web app name prefix for the user name).

The browser shows the root folder of the web app.

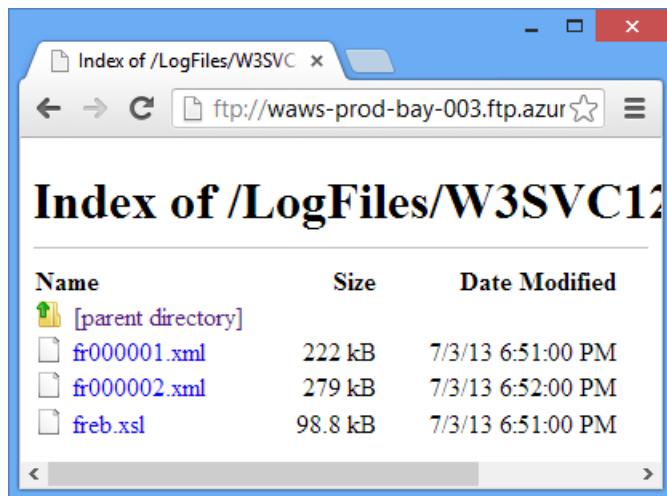
7. Open the *LogFiles* folder.



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.

The screenshot shows the Request Diagnostics interface for a GET request to `http://example1z.azurewebsites.net:80/Home/Contactx`. The top section displays summary information including Site ID (1261305712), Process ID (3268), Failure Reason (STATUS_CODE), Trigger Status (404), Final Status (404), and Time Taken (5125 msec). Below this, detailed request metadata is shown: Url (`http://example1z.azurewebsites.net:80/Home/Contactx`), App Pool (example1z), Authentication (anonymous), User from token (IIS APPPOOL\example1z), and Activity ID ({00000000-0000-0000-7369-0180000000F1}).

The Errors & Warnings section lists a single trace event (No. 191) with Severity Warning. The event details show it occurred at `- MODULE_SET_RESPONSE_ERROR_STATUS` managed by `ManagedPipelineHandler`. The event properties include ModuleName (`ManagedPipelineHandler`), Notification (`EXECUTE_REQUEST_HANDLER`), HttpStatus (404), HttpReason (Not Found), HttpSubStatus (0), ErrorCode (The operation completed successfully. (0x0)), and ConfigExceptionInfo.

A link "See all events for the request" is provided. The bottom section shows a table of tracing events:

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 Azure web app. The following sections provide links to more resources on related topics:

- Azure web app 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

Azure web app troubleshooting

For more information about troubleshooting web apps in Azure App Service, see the following resources:

- [How to monitor web apps](#)
- [Investigating Memory Leaks in Azure Web Apps with Visual Studio 2013](#). Microsoft ALM blog post about Visual Studio features for analyzing managed memory issues.
- [Azure web apps 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 Azure web apps and WebJobs, see the following resources:

- [Introduction to Remote Debugging Azure App Service Web Apps](#).
- [Introduction to Remote Debugging Azure App Service Web Apps part 2 - Inside Remote debugging](#)
- [Introduction to Remote Debugging on Azure App Service Web Apps part 3 - Multi-Instance environment and GIT](#)
- [WebJobs Debugging \(video\)](#)

If your web 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 Azure web 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 web 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 Web Apps.

Best practices and troubleshooting guide for node applications on Azure Web Apps

11/10/2017 • 13 min to read • [Edit Online](#)

In this article, you learn best practices and troubleshooting steps for [node applications](#) running on Azure Web Apps (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 setting this 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 Web Apps, the default value for this is Infinite. You don't have to worry about this setting. Outside Azure Web Apps, the default value is 1024. You can configure this 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 request over 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 Web Apps.
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 Web Apps 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 websockets. You must enable websockets on your Azure webapp to use this version. See <http://www.ranjithr.com/?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>
```

Enabling 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), so it is best to scope this setting only to endpoints that require response streaming (for example, using the 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

This is a semi-colon separated list of files that are watched for changes. A change to a file causes the application to recycle. Each entry consists of an optional directory name plus a required file name that 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;web.config”

- 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 lot of memory and want to page out memory to disk occasionally to free up some 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 gleaned by looking at the URL fragment, but a much better visualization is achieved by opening the URL in the browser.

- `loggingEnabled`

This setting controls the logging of `stdout` and `stderr` by iisnode. lisnode 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. lisnode is integrated with node-inspector. By enabling this setting, you enable debugging of your node application. Once this setting is enabled, iisnode will lay out the necessary 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 this to a GUID, for example, so that it is more difficult to be discovered by others.

Check this [link](#) for more details on debugging.

Scenarios and recommendations/troubleshooting

My node application is making too many 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. For example, you could use the `agentkeepalive` module as your keep alive agent when making these outbound calls. This makes sure that the sockets are reused on your Azure webapp VM and reducing the overhead of creating new sockets for every outbound request. Also, this makes sure that you are using less number of sockets to make many outbound requests and therefore you don't exceed the `maxSockets` that are allocated per VM. The recommendation on Azure Web Apps is to set the `agentKeepAlive maxSockets` value to a total of 160 sockets per VM. This means that if you have four node.exe running on the VM, you want to set the `agentKeepAlive maxSockets` to 40 per node.exe, which is 160 total per VM.

Example `agentKeepALive` configuration:

```
var keepaliveAgent = new Agent({
  maxSockets: 40,
  maxFreeSockets: 10,
  timeout: 60000,
  keepAliveTimeout: 300000
});
```

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 will probably get a recommendation from Azure Web Apps 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. In cases where you think 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 Web Apps with V8-Profiler

For example, let's say you have a hello world app that you want to profile as follows:

```
var http = require('http');
function WriteConsoleLog() {
  for(var 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 your scm site <https://yoursite.scm.azurewebsites.net/DebugConsole>

You see a command prompt as shown below. Go into your site/wwwroot directory.

... / wwwroot + | 4 items   

	Name	Modified	Size
  	node_modules	5/25/2016, 12:53:41 PM	
  	hostingstart.html	5/25/2016, 12:51:16 PM	198 KB
  	server.js	5/25/2016, 12:57:09 PM	1 KB
  	web.config	5/25/2016, 12:52:34 PM	1 KB

▼ ▲ [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>npm install v8-profiler
```

Run the command "npm install v8-profiler".

This should install v8-profiler under node_modules directory and all of its dependencies. Now, edit your server.js to profile your application.

```
var http = require('http');
var profiler = require('v8-profiler');
var fs = require('fs');

function WriteConsoleLog() {
    for(var 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.

	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

▼ ▲

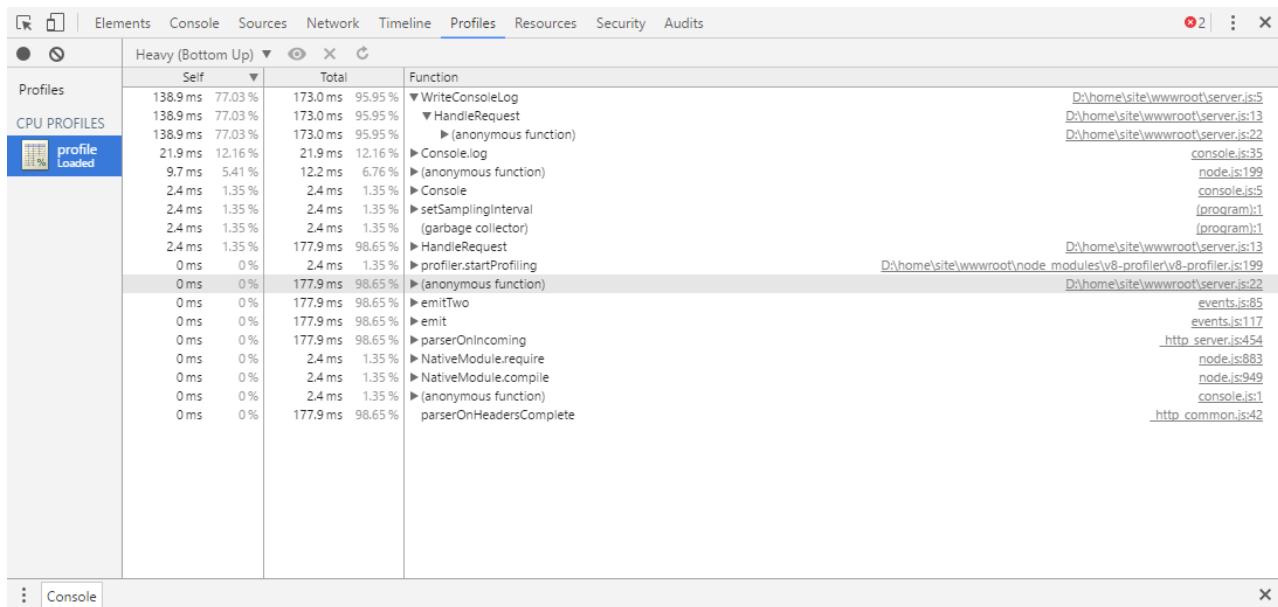
Use old console

```
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. This 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 Web Apps 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.

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 this could be happening:

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 so you can fix your application based on this.
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 to let other processes get a chance to do some work. To fix this, either make sure your application is not leaking memory OR if your application needs to use a great deal of memory, scale up to a larger VM with a lot more RAM.

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 – Please check d:\home\LogFiles\Application\logging-errors.txt file for the details on the exception thrown. This file has the stack trace so you can fix your application based on this.

My node application takes too much time to start (Cold Start)

The most common reason for an application taking too long to start 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 reside on the network share on Azure Web Apps, 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. This means that the call to require('module') should be made when you actually need it within the function you try when using the module.
3. Azure Web Apps 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

This [source file](#) lists all of the possible status/substatus combinations iisnode can return in case of 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?
-------------	----------------	------------------

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 – You should never see this but if you do, 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.
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 – Check if the node is consuming excessive CPU

There is a setting within NODE.exe called NODE_PENDING_PIPE_INSTANCES. By default, outside of Azure Web Apps, this value is 4. This means that node.exe can only accept four requests at a time on the named pipe. On Azure Web Apps, this value is set to 5000. This value should be good enough for most node applications running on Azure Web Apps. You should not see 503.1003 on Azure Web Apps 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 your Azure web apps

9/19/2017 • 4 min to read • [Edit Online](#)

"502 bad gateway" and "503 service unavailable" are common errors in your web 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 web 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 Web Apps](#) 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](#).

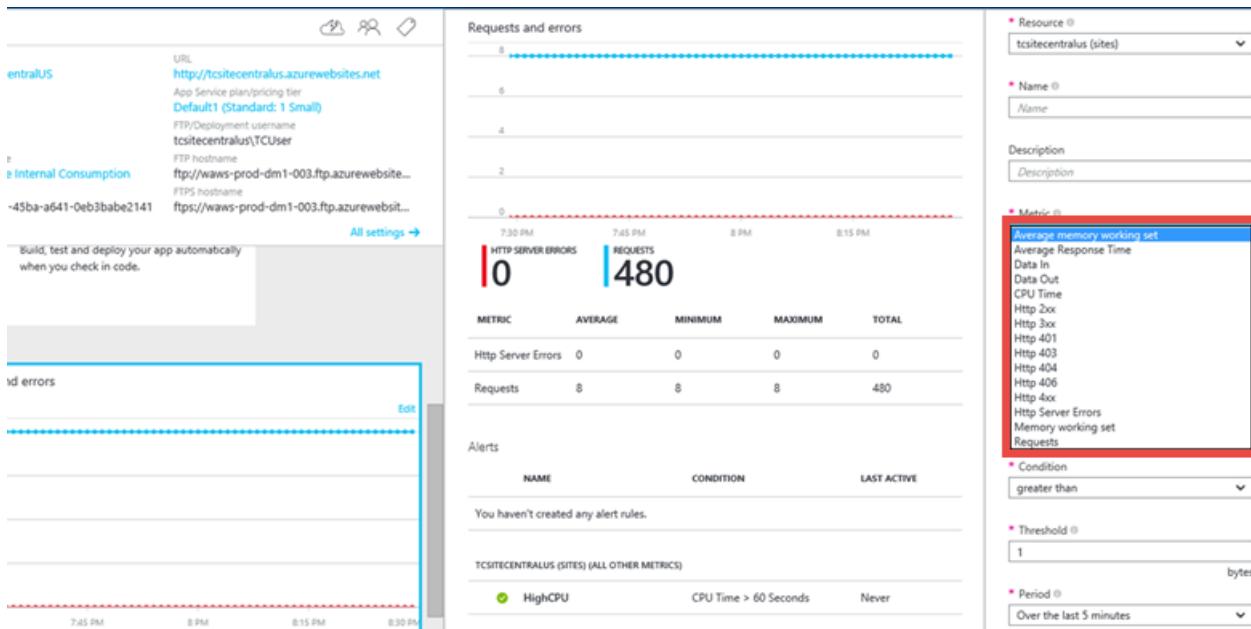
Monitor your web app

This option enables you to find out if your application is having any issues. In your web 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 web app are

- Average memory working set
- Average response time
- CPU time
- Memory working set

- Requests



For more information, see:

- [Monitor Web Apps in Azure App Service](#)
- [Receive alert notifications](#)

2. Collect data

Use the Azure App Service Support Portal

Web Apps provides you with the ability to troubleshoot issues related to your web app by looking at HTTP logs, event logs, process dumps, and more. You can access all this information using our Support portal at <http://<your app name>.scm.azurewebsites.net/Support>

The Azure App Service Support portal provides you with three separate tabs to support the three steps of a common troubleshooting scenario:

1. Observe current behavior
2. Analyze by collecting diagnostics information and running the built-in analyzers
3. Mitigate

If the issue is happening right now, click **Analyze > Diagnostics > Diagnose Now** to create a diagnostic session for you, which will collect HTTP logs, event viewer logs, memory dumps, PHP error logs and PHP process report.

Once the data is collected, it will also run an analysis on the data and provide you with an HTML report.

In case you want to download the data, by default, it would be stored in the D:\home\data\Daas folder.

For more information on the Azure App Service Support portal, see [New Updates to Support Site Extension for Azure Websites](#).

Use the Kudu Debug Console

Web Apps 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 web 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 web app.

For more information on features available in Kudu, see [Azure Websites online tools you should know about](#).

3. Mitigate the issue

Scale the web app

In Azure App Service, for increased performance and throughput, you can adjust the scale at which you are running your application. Scaling up a web 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 a web 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 web 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 web 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 web app

This is often the simplest way to recover from one-time issues. On the [Azure Portal](#), on your web app's blade, you have the options to stop or restart your app.



You can also manage your web app using Azure Powershell. For more information, see [Using Azure PowerShell with Azure Resource Manager](#).

Troubleshoot slow web app performance issues in Azure App Service

9/19/2017 • 9 min to read • [Edit Online](#)

This article helps you troubleshoot slow web 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 web 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 Web Apps](#) gives you various options at each step.

1. Observe and monitor application behavior

Track Service health

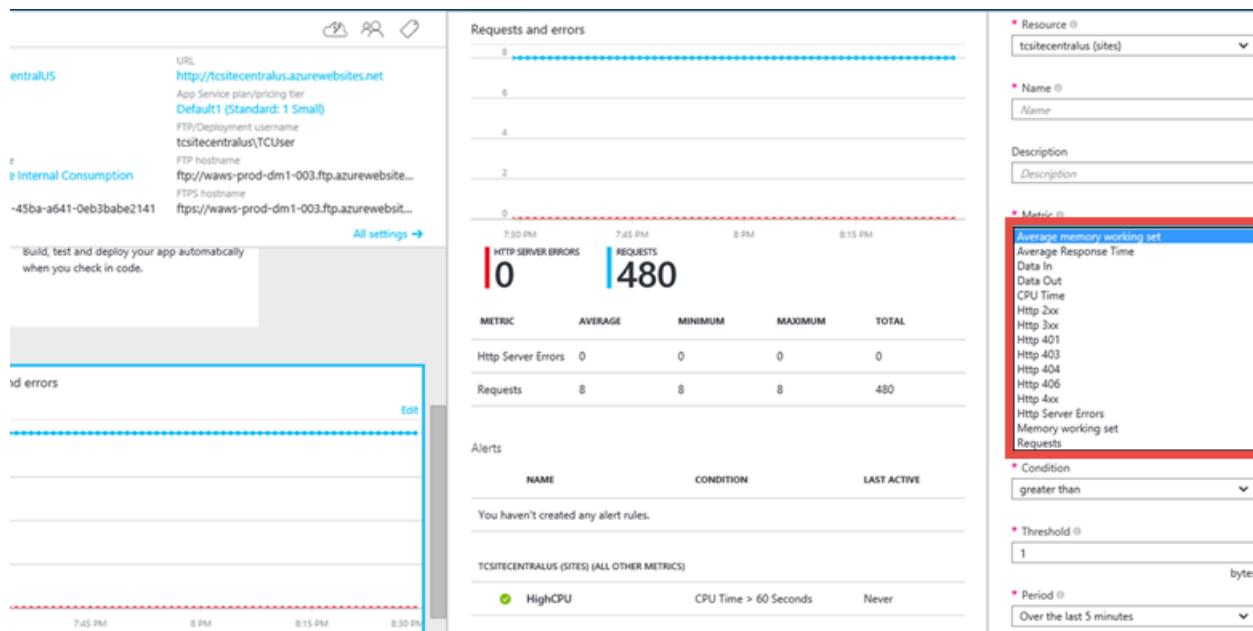
Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the [Azure portal](#). For more information, see [Track service health](#).

Monitor your web app

This option enables you to find out if your application is having any issues. In your web 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 web app are

- Average memory working set
- Average response time
- CPU time
- Memory working set
- Requests



For more information, see:

- [Monitor Web Apps in Azure App Service](#)
- [Receive alert notifications](#)

Monitor web endpoint status

If you are running your web app in the **Standard** pricing tier, Web Apps 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 *site extensions*.

Each App Service web app provides an extensible management endpoint that allows you to use a powerful set of tools deployed as site extensions. Extensions include:

- Source code editors like [Visual Studio Team Services](#).
- Management tools for connected resources such as a MySQL database connected to a web app.

[Azure Application Insights](#) and [New Relic](#) are two of the performance monitoring site extensions that are available.

To use New Relic, you install an agent at runtime. To use Azure Application Insights, you rebuild your code with an SDK, and 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.

To use Application Insights, see [Monitor performance in web applications](#).

To use New Relic, see [New Relic Application Performance Management on Azure](#).

2. Collect data

The Web Apps environment 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 web 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 web 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 Web Apps, 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 web 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 Azure web apps with Application Insights](#).

Use Remote Profiling

In Azure App Service, Web Apps, API Apps, and WebJobs can be remotely profiled. Choose this option if you have access to the web 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 web apps in Azure App Service](#).

Use the Azure App Service support portal

Web Apps provides you with the ability to troubleshoot issues related to your web app by looking at HTTP logs, event logs, process dumps, and more. You can access all this information using our Support portal at <http://<your app name>.scm.azurewebsites.net/Support>

The Azure App Service support portal provides you with three separate tabs to support the three steps of a common troubleshooting scenario:

1. Observe current behavior
2. Analyze by collecting diagnostics information and running the built-in analyzers
3. Mitigate

If the issue is happening right now, click **Analyze > Diagnostics > Diagnose Now** to create a diagnostic session for you, which collects HTTP logs, event viewer logs, memory dumps, PHP error logs, and PHP process report.

Once the data is collected, the support portal runs an analysis on the data and provides you with an HTML report.

In case you want to download the data, by default, it would be stored in the D:\home\data\Daas folder.

For more information on the Azure App Service support portal, see [New Updates to Support Site Extension for Azure Websites](#).

Use the Kudu Debug Console

Web Apps 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 web 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 web app.

For more information on features available in Kudu, see [Azure Websites Team Services tools you should know about](#).

3. Mitigate the issue

Scale the web app

In Azure App Service, for increased performance and throughput, you can adjust the scale at which you are running your application. Scaling up a web 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 a web 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 web 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 web 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 web app

Restarting is often the simplest way to recover from one-time issues. On the [Azure portal](#), on your web app's blade, you have the options to stop or restart your app.



You can also manage your web app using Azure Powershell. For more information, see [Using Azure PowerShell with Azure Resource Manager](#).

Application performance FAQs for Web Apps in Azure

1/2/2018 • 8 min to read • [Edit Online](#)

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 the 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 [AzureRM.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 turned 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 240 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

11/3/2017 • 4 min 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 the 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?

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'. Please 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 2017.

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 are 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 Visual Studio Team Services, 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?

To resolve FTP issues:

1. Verify that you are 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 **Settings > App Deployment**.

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 Visual Studio Team Services code with App Service?

You have two options for using continuous deployment with Visual Studio Team Services:

- Use a Git project. Connect via App Service by using the deployment options for that repo.
- 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 a Visual Studio Team Services account 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

11/22/2017 • 8 min 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 the 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**.

My ClearDB database is down. How do I resolve this?

For database-related issues, contact [ClearDB support](#).

For answers to common questions about ClearDB, see [ClearDB FAQs](#).

Why wasn't my ClearDB database migrated during my subscription migration?

When you perform resource migration across subscriptions, some limitations apply. A ClearDB MySQL database is a third-party service and is not migrated during an Azure subscription migration.

If you don't manage the migration of your MySQL database before you migrate your Azure resources, your ClearDB MySQL database might be unavailable. To avoid this, first, manually migrate your ClearDB database, and then migrate the Azure subscription for your web app.

For more information, see [FAQs for ClearDB MySQL databases with Azure App Service](#).

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); //Supress errors and warnings to screendefine('WP_DEBUG_DISPLAY', false); //Supress 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?

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>
<environmentVariable name="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 Edge, where the issue does not occur.

Configuration and management FAQs for Web Apps in Azure

11/22/2017 • 15 min 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 the 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 [Bind an existing custom SSL certificate to an Azure web 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**.

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.

If your website is hosted in App Service Environment for PowerApps, to learn how to get your outbound IP address, 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?

App Service certificates are considered Azure resources. They are not intended to use outside your Azure services. You cannot 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 or edit a URL rewrite rule?

To add or edit a URL rewrite rule:

1. Set up Internet Information Services (IIS) Manager so that it connects to your App Service web app. To learn how to connect IIS Manager to App Service, see [Remote administration of Azure websites by using IIS Manager](#).
2. In IIS Manager, add or edit a URL rewrite rule. To learn how to add or edit a URL rewrite rule, see [Create rewrite rules for the URL rewrite module](#).

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 4016, 4018, and 4020 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 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?

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.

```
 Login-AzureRmAccount  
 Set-AzureRmContext -SubscriptionId <subId>  
 $actionProperties = @{  
     "Name"= "<Customer Email Address>"  
 };  
 Invoke-AzureRmResourceAction -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 identity 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](#).