

Contents

Overview

- Developer guide

- SDKs and tools

Quickstart

- Web Apps

- Virtual machines

 - Linux

 - Windows

- Serverless

- Microservices

 - Service Fabric

 - Container Service

 - Azure Spring Cloud

Tutorials

- Create and deploy a web app

 - .NET with SQL DB

 - Node.js with Mongo DB

 - PHP with MySQL

 - Java with MySQL

- Deploy complex VM templates

 - Linux

 - Windows

- Create an Azure connected function

- Docker deploy web app on Linux

Samples

- Azure CLI

 - Web Apps

 - Linux VM

 - Windows VM

Azure PowerShell

Web Apps

Linux VM

Windows VM

Concepts

Billing and subscriptions

Hosting comparisons

What is App Service?

Virtual machines

Linux VMs

Windows VMs

Service Fabric overview

How to guides

Plan

Web application architectures

VM architectures

Connect to on-premises networks

Microservices patterns/scenarios

Develop

Linux VM

Windows VM

Serverless apps

Microservices cluster

Deploy

Web and mobile apps from source control

Microservices locally

Linux VM

Windows VM

Store data

Blobs

File shares

Key-value pairs

JSON documents

Relational tables

Message queues

Scale

Web and mobile apps

Virtual machines

Microservice apps

Secure

Web and mobile apps

Backup

Web and mobile apps

Virtual machines

Monitor

Web and mobile apps

Windows VM

Microservices

Billing alerts

Automate

Scale Linux VM

Scale Windows VM

Reference

REST

SDKs

.NET

Java

Node.js

PHP

Python

Ruby

Command line interfaces

Azure CLI

Azure PowerShell

[Billing](#)

[Resources](#)

[Azure limits and quotas](#)

[Azure regions](#)

[Azure Roadmap](#)

[Pricing calculator](#)

[Samples](#)

[Videos](#)

Get started guide for Azure developers

6/10/2021 • 21 minutes to read • [Edit Online](#)

What is Azure?

Azure is a complete cloud platform that can host your existing applications and streamline new application development. Azure can even enhance on-premises applications. Azure integrates the cloud services that you need to develop, test, deploy, and manage your applications, all while taking advantage of the efficiencies of cloud computing.

By hosting your applications in Azure, you can start small and easily scale your application as your customer demand grows. Azure also offers the reliability that's needed for high-availability applications, even including failover between different regions. The [Azure portal](#) lets you easily manage all your Azure services. You can also manage your services programmatically by using service-specific APIs and templates.

This guide is an introduction to the Azure platform for application developers. It provides guidance and direction that you need to start building new applications in Azure or migrating existing applications to Azure.

Where do I start?

With all the services that Azure offers, it can be an intimidating task to figure out which services you need to support your solution architecture. This section highlights the Azure services that developers commonly use. For a list of all Azure services, see the [Azure documentation](#).

First, you must decide on how to host your application in Azure. Do you need to manage your entire infrastructure as a virtual machine (VM)? Can you use the platform management facilities that Azure provides? Maybe you need a serverless framework to host code execution only?

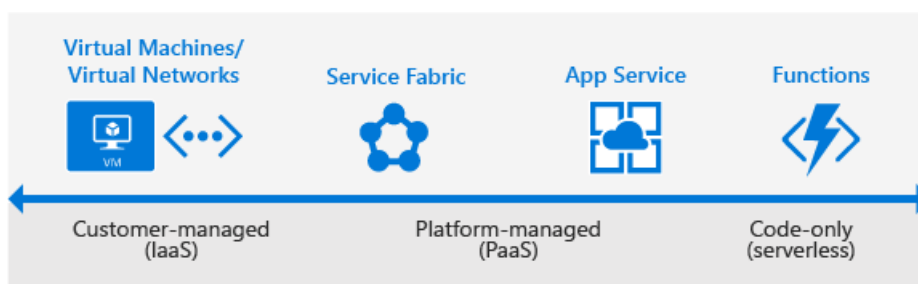
Your application needs cloud storage, which Azure provides several options for. You can take advantage of Azure's enterprise authentication. There are also tools for cloud-based development and monitoring, and most hosting services offer DevOps integration.

Now, let's look at some of the specific services that we recommend investigating for your applications.

Application hosting

Azure provides several cloud-based compute offerings to run your application so that you don't have to worry about the infrastructure details. You can easily scale up or scale out your resources as your application usage grows.

Azure offers services that support your application development and hosting needs. Azure provides Infrastructure as a Service (IaaS) to give you full control over your application hosting. Azure's Platform as a Service (PaaS) offerings provide the fully managed services needed to power your apps. There's even true serverless hosting in Azure where all you need to do is write your code.



Azure App Service

When you want the quickest path to publish your web-based projects, consider Azure App Service. App Service makes it easy to extend your web apps to support your mobile clients and publish easily consumed REST APIs. This platform provides authentication by using social providers, traffic-based autoscaling, testing in production, and continuous and container-based deployments.

You can create web apps, mobile app back ends, and API apps.

Because all three app types share the App Service runtime, you can host a website, support mobile clients, and expose your APIs in Azure, all from the same project or solution. To learn more about App Service, see [What is Azure Web Apps](#).

App Service has been designed with DevOps in mind. It supports various tools for publishing and continuous integration deployments. These tools include GitHub webhooks, Jenkins, Azure DevOps, TeamCity, and others.

You can migrate your existing applications to App Service by using the [online migration tool](#).

When to use: Use App Service when you're migrating existing web applications to Azure, and when you need a fully-managed hosting platform for your web apps. You can also use App Service when you need to support mobile clients or expose REST APIs with your app.

Get started: App Service makes it easy to create and deploy your first [web app](#), [mobile app](#), or [API app](#).

Try it now: App Service lets you provision a short-lived app to try the platform without having to sign up for an Azure account. Try the platform and [create your Azure App Service app](#).

Azure Virtual Machines

As an Infrastructure as a Service (IaaS) provider, Azure lets you deploy to or migrate your application to either Windows or Linux VMs. Together with Azure Virtual Network, Azure Virtual Machines supports the deployment of Windows or Linux VMs to Azure. With VMs, you have total control over the configuration of the machine. When using VMs, you're responsible for all server software installation, configuration, maintenance, and operating system patches.

Because of the level of control that you have with VMs, you can run a wide range of server workloads on Azure that don't fit into a PaaS model. These workloads include database servers, Windows Server Active Directory, and Microsoft SharePoint. For more information, see the Virtual Machines documentation for either [Linux](#) or [Windows](#).

When to use: Use Virtual Machines when you want full control over your application infrastructure or to migrate on-premises application workloads to Azure without having to make changes.

Get started: Create a [Linux VM](#) or [Windows VM](#) from the Azure portal.

Azure Functions (serverless)

Rather than worrying about building out and managing a whole application or the infrastructure to run your code, what if you could just write your code and have it run in response to events or on a schedule? [Azure Functions](#) is a "serverless"-style offering that lets you write just the code you need. With Functions, you can trigger code execution with HTTP requests, webhooks, cloud service events, or on a schedule. You can code in your development language of choice, such as C#, F#, Node.js, Python, or PHP. With consumption-based billing, you pay only for the time that your code executes, and Azure scales as needed.

When to use: Use Azure Functions when you have code that is triggered by other Azure services, by web-based events, or on a schedule. You can also use Functions when you don't need the overhead of a complete hosted project or when you only want to pay for the time that your code runs. To learn more, see [Azure Functions Overview](#).

Get started: Follow the Functions quickstart tutorial to [create your first function](#) from the portal.

Try it now: Azure Functions lets you run your code without having to sign up for an Azure account. Try it now at and [create your first Azure Function](#).

Azure Service Fabric

Azure Service Fabric is a distributed systems platform. This platform makes it easy to build, package, deploy, and manage scalable and reliable microservices. It also provides comprehensive application management capabilities such as:

- Provisioning
- Deploying
- Monitoring
- Upgrading/patching
- Deleting

Apps, which run on a shared pool of machines, can start small and scale to hundreds or thousands of machines as needed.

Service Fabric supports WebAPI with Open Web Interface for .NET (OWIN) and ASP.NET Core. It provides SDKs for building services on Linux in both .NET Core and Java. To learn more about Service Fabric, see the [Service Fabric documentation](#).

When to use: Service Fabric is a good choice when you're creating an application or rewriting an existing application to use a microservice architecture. Use Service Fabric when you need more control over, or direct access to, the underlying infrastructure.

Get started: [Create your first Azure Service Fabric application](#).

Azure Spring Cloud

Azure Spring Cloud is a serverless microservices platform that enables you to build, deploy, scale and monitor your applications in the cloud. Use Spring Cloud to bring modern microservice patterns to Spring Boot apps, eliminating boilerplate code to quickly build robust Java apps.

- Leverage managed versions of Spring Cloud Service Discovery and Config Server, while we ensure those critical components are running in optimum conditions.
- Focus on building your business logic and we will take care of your service runtime with security patches, compliance standards and high availability.
- Manage application lifecycle (e.g.: deploy, start, stop, scale) on top of Azure Kubernetes Service.
- Easily bind connections between your apps and Azure services such as Azure Database for MySQL and Azure Cache for Redis.
- Monitor and troubleshoot microservices and applications using enterprise-grade unified monitoring tools that offer deep insights on application dependencies and operational telemetry.

When to use: As a fully managed service Azure Spring Cloud is a good choice when you're minimizing operational cost running Spring Boot/Spring Cloud based microservices on Azure.

Get started: [Deploy your first Azure Spring Cloud application](#).

Enhance your applications with Azure services

Along with application hosting, Azure provides service offerings that can enhance the functionality. Azure can also improve the development and maintenance of your applications, both in the cloud and on-premises.

Hosted storage and data access

Most applications must store data, so however you decide to host your application in Azure, consider one or more of the following storage and data services.

- **Azure Cosmos DB:** A globally distributed, multi-model database service. This database enables you to elastically scale throughput and storage across any number of geographical regions with a comprehensive SLA.

When to use: When your application needs document, table, or graph databases, including MongoDB databases, with multiple well-defined consistency models.

Get started: [Build an Azure Cosmos DB web app](#). If you're a MongoDB developer, see [Build a MongoDB web app with Azure Cosmos DB](#).

- **Azure Storage:** Offers durable, highly available storage for blobs, queues, files, and other kinds of nonrelational data. Storage provides the storage foundation for VMs.

When to use: When your app stores nonrelational data, such as key-value pairs (tables), blobs, files shares, or messages (queues).

Get started: Choose from one of these types storage: [blobs](#), [tables](#), [queues](#), or [files](#).

- **Azure SQL Database:** An Azure-based version of the Microsoft SQL Server engine for storing relational tabular data in the cloud. SQL Database provides predictable performance, scalability with no downtime, business continuity, and data protection.

When to use: When your application requires data storage with referential integrity, transactional support, and support for TSQL queries.

Get started: [Create a database in Azure SQL Database in minutes by using the Azure portal](#).

You can use [Azure Data Factory](#) to move existing on-premises data to Azure. If you aren't ready to move data to the cloud, [Hybrid Connections](#) in Azure App Service lets you connect your App Service hosted app to on-premises resources. You can also connect to Azure data and storage services from your on-premises applications.

Docker support

Docker containers, a form of OS virtualization, let you deploy applications in a more efficient and predictable way. A containerized application works in production the same way as on your development and test systems. You can manage containers by using standard Docker tools. You can use your existing skills and popular open-source tools to deploy and manage container-based applications on Azure.

Azure provides several ways to use containers in your applications.

- **Azure Kubernetes Service:** Lets you create, configure, and manage a cluster of virtual machines that are preconfigured to run containerized applications. To learn more about Azure Kubernetes Service, see [Azure Kubernetes Service introduction](#).

When to use: When you need to build production-ready, scalable environments that provide additional scheduling and management tools, or when you're deploying a Docker Swarm cluster.

Get started: [Deploy a Kubernetes Service cluster](#).

- **Docker Machine:** Lets you install and manage a Docker Engine on virtual hosts by using docker-machine commands.

When to use: When you need to quickly prototype an app by creating a single Docker host.

- **Custom Docker image for App Service:** Lets you use Docker containers from a container registry or a customer container when you deploy a web app on Linux.

When to use: When deploying a web app on Linux to a Docker image.

Get started: [Use a custom Docker image for App Service on Linux.](#)

Authentication

It's crucial to not only know who is using your applications, but also to prevent unauthorized access to your resources. Azure provides several ways to authenticate your app clients.

- **Azure Active Directory (Azure AD):** The Microsoft multitenant, cloud-based identity and access management service. You can add single-sign on (SSO) to your applications by integrating with Azure AD. You can access directory properties by using the Azure AD Graph API directly or the Microsoft Graph API. You can integrate with Azure AD support for the OAuth2.0 authorization framework and Open ID Connect by using native HTTP/REST endpoints and the multiplatform Azure AD authentication libraries.

When to use: When you want to provide an SSO experience, work with Graph-based data, or authenticate domain-based users.

Get started: To learn more, see the [Azure Active Directory developer's guide.](#)

- **App Service Authentication:** When you choose App Service to host your app, you also get built-in authentication support for Azure AD, along with social identity providers—including Facebook, Google, Microsoft, and Twitter.

When to use: When you want to enable authentication in an App Service app by using Azure AD, social identity providers, or both.

Get started: To learn more about authentication in App Service, see [Authentication and authorization in Azure App Service.](#)

To learn more about security best practices in Azure, see [Azure security best practices and patterns.](#)

Monitoring

With your application up and running in Azure, you need to monitor performance, watch for issues, and see how customers are using your app. Azure provides several monitoring options.

- **Application Insights:** An Azure-hosted extensible analytics service that integrates with Visual Studio to monitor your live web applications. It gives you the data that you need to improve the performance and usability of your apps continuously. This improvement occurs whether you host your applications on Azure or not.

Get started: Follow the [Application Insights tutorial.](#)

- **Azure Monitor:** A service that helps you to visualize, query, route, archive, and act on the metrics and logs that you generate with your Azure infrastructure and resources. Monitor is a single source for monitoring Azure resources and provides the data views that you see in the Azure portal.

Get started: [Get started with Azure Monitor.](#)

DevOps integration

Whether it's provisioning VMs or publishing your web apps with continuous integration, Azure integrates with most of the popular DevOps tools. You can work with the tools that you already have and maximize your

existing experience with support for tools like:

- Jenkins
- GitHub
- Puppet
- Chef
- TeamCity
- Ansible
- Azure DevOps

Get started: To see DevOps options for an App Service app, see [Continuous Deployment to Azure App Service](#).

Try it now: [Try out several of the DevOps integrations](#).

Azure regions

Azure is a global cloud platform that is generally available in many regions around the world. When you provision a service, application, or VM in Azure, you're asked to select a region. This region represents a specific datacenter where your application runs or where your data is stored. These regions correspond to specific locations, which are published on the [Azure regions](#) page.

Choose the best region for your application and data

One of the benefits of using Azure is that you can deploy your applications to various datacenters around the globe. The region that you choose can affect the performance of your application. For example, it's better to choose a region that's closer to most of your customers to reduce latency in network requests. You might also want to select your region to meet the legal requirements for distributing your app in certain countries/regions. It's always a best practice to store application data in the same datacenter or in a datacenter as near as possible to the datacenter that is hosting your application.

Multi-region apps

Although unlikely, it's not impossible for an entire datacenter to go offline because of an event such as a natural disaster or Internet failure. It's a best practice to host vital business applications in more than one datacenter to provide maximum availability. Using multiple regions can also reduce latency for global users and provide additional opportunities for flexibility when updating applications.

Some services, such as Virtual Machine and App Services, use [Azure Traffic Manager](#) to enable multi-region support with failover between regions to support high-availability enterprise applications. For an example, see [Azure reference architecture: Run a web application in multiple regions](#).

When to use: When you have enterprise and high-availability applications that benefit from failover and replication.

How do I manage my applications and projects?

Azure provides a rich set of experiences for you to create and manage your Azure resources, applications, and projects—both programmatically and in the [Azure portal](#).

Command-line interfaces and PowerShell

Azure provides two ways to manage your applications and services from the command line. You can use tools like Bash, Terminal, the command prompt, or your command-line tool of choice. Usually, you can do the same tasks from the command line as in the Azure portal—such as creating and configuring virtual machines, virtual networks, web apps, and other services.

- [Azure Command-Line Interface \(CLI\)](#): Lets you connect to an Azure subscription and program various tasks against Azure resources from the command line.
- [Azure PowerShell](#): Provides a set of modules with cmdlets that enable you to manage Azure resources by using Windows PowerShell.

Azure portal

The [Azure portal](#) is a web-based application. You can use the Azure portal to create, manage, and remove Azure resources and services. It includes:

- A configurable dashboard
- Azure resource management tools
- Access to subscription settings and billing information. For more information, see the [Azure portal overview](#).

REST APIs

Azure is built on a set of REST APIs that support the Azure portal UI. Most of these REST APIs are also supported to let you programmatically provision and manage your Azure resources and applications from any Internet-enabled device. For the complete set of REST API documentation, see the [Azure REST SDK reference](#).

APIs

Along with REST APIs, many Azure services also let you programmatically manage resources from your applications by using platform-specific Azure SDKs, including SDKs for the following development platforms:

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

Services such as [Mobile Apps](#) and [Azure Media Services](#) provide client-side SDKs to let you access services from web and mobile client apps.

Azure Resource Manager

Running your app on Azure likely involves working with multiple Azure services. These services follow the same life cycle and can be thought of as a logical unit. For example, a web app might use Web Apps, SQL Database, Storage, Azure Cache for Redis, and Azure Content Delivery Network services. [Azure Resource Manager](#) lets you work with the resources in your application as a group. You can deploy, update, or delete all the resources in a single, coordinated operation.

Along with logically grouping and managing related resources, Azure Resource Manager includes deployment capabilities that let you customize the deployment and configuration of related resources. For example, you can use Resource Manager deploy and configure an application. This application can consist of multiple virtual machines, a load balancer, and a database in Azure SQL Database as a single unit.

You develop these deployments by using an Azure Resource Manager template, which is a JSON-formatted document. Templates let you define a deployment and manage your applications by using declarative templates, rather than scripts. Your templates can work for different environments, such as testing, staging, and production. For example, you can use templates to add a button to a GitHub repo that deploys the code in the repo to a set of Azure services with a single click.

When to use: Use Resource Manager templates when you want a template-based deployment for your app that you can manage programmatically by using REST APIs, the Azure CLI, and Azure PowerShell.

Understanding accounts, subscriptions, and billing

As developers, we like to dive right into the code and try to get started as fast as possible with making our applications run. We certainly want to encourage you to start working in Azure as easily as possible. To help make it easy, Azure offers a [free trial](#). Some services even have a "Try it for free" functionality, like [Azure App Service](#), which doesn't require you to even create an account. As fun as it is to dive into coding and deploying your application to Azure, it's also important to take some time to understand how Azure works. Specifically, you should understand how it works from a standpoint of user accounts, subscriptions, and billing.

What is an Azure account?

To create or work with an Azure subscription, you must have an Azure account. An Azure account is simply an identity in Azure AD or in a directory, such as a work or school organization, that Azure AD trusts. If you don't belong to such an organization, you can always create a subscription by using your Microsoft Account, which is trusted by Azure AD. To learn more about integrating on-premises Windows Server Active Directory with Azure AD, see [Integrating your on-premises identities with Azure Active Directory](#).

Every Azure subscription has a trust relationship with an Azure AD instance. This means that it trusts that directory to authenticate users, services, and devices. Multiple subscriptions can trust the same directory, but a subscription trusts only one directory. To learn more, see [How Azure subscriptions are associated with Azure Active Directory](#).

As well as defining individual Azure account identities, also called *users*, you can define *groups* in Azure AD. Creating user groups is a good way to manage access to resources in a subscription by using role-based access control (RBAC). To learn how to create groups, see [Create a group in Azure Active Directory preview](#). You can also create and manage groups by [using PowerShell](#).

Manage your subscriptions

A subscription is a logical grouping of Azure services that is linked to an Azure account. A single Azure account can contain multiple subscriptions. Billing for Azure services is done on a per-subscription basis. For a list of the available subscription offers by type, see [Microsoft Azure Offer Details](#). Azure subscriptions have an Account Administrator who has full control over the subscription. They also have a Service Administrator who has control over all services in the subscription. For information about classic subscription administrators, see [Add or change Azure subscription administrators](#). Individual accounts can be granted detailed control of Azure resources using [Azure role-based access control \(Azure RBAC\)](#).

Resource groups

When you provision new Azure services, you do so in a given subscription. Individual Azure services, which are also called resources, are created in the context of a resource group. Resource groups make it easier to deploy and manage your application's resources. A resource group should contain all the resources for your application that you want to work with as a unit. You can move resources between resource groups and even to different subscriptions. To learn about moving resources, see [Move resources to new resource group or subscription](#).

The Azure Resource Explorer is a great tool for visualizing the resources that you've already created in your subscription. To learn more, see [Use Azure Resource Explorer to view and modify resources](#).

Grant access to resources

When you allow access to Azure resources, it's always a best practice to provide users with the least privilege that's required to do a given task.

- **Azure role-based access control (Azure RBAC):** In Azure, you can grant access to user accounts (principals) at a specified scope: subscription, resource group, or individual resources. Azure RBAC lets you deploy resources into a resource group and grant permissions to a specific user or group. It also lets you limit access to only the resources that belong to the target resource group. You can also grant access

to a single resource, such as a virtual machine or virtual network. To grant access, you assign a role to the user, group, or service principal. There are many predefined roles, and you can also define your own custom roles. To learn more, see [What is Azure role-based access control \(Azure RBAC\)?](#).

When to use: When you need fine-grained access management for users and groups or when you need to make a user an owner of a subscription.

Get started: To learn more, see [Assign Azure roles using the Azure portal](#).

- **Service principal objects:** Along with providing access to user principals and groups, you can grant the same access to a service principal.

When to use: When you're programmatically managing Azure resources or granting access for applications. For more information, see [Create Active Directory application and service principal](#).

Tags

Azure Resource Manager lets you assign custom tags to individual resources. Tags, which are key-value pairs, can be helpful when you need to organize resources for billing or monitoring. Tags provide you a way to track resources across multiple resource groups. You can assign tags the following ways:

- In the portal
- In the Azure Resource Manager template
- Using the REST API
- Using the Azure CLI
- Using PowerShell

You can assign multiple tags to each resource. To learn more, see [Using tags to organize your Azure resources](#).

Billing

In the move from on-premises computing to cloud-hosted services, tracking and estimating service usage and related costs are significant concerns. It's important to estimate what new resources cost to run on a monthly basis. You can also project how the billing looks for a given month based on the current spending.

Get resource usage data

Azure provides a set of Billing REST APIs that give access to resource consumption and metadata information for Azure subscriptions. These Billing APIs give you the ability to better predict and manage Azure costs. You can track and analyze spending in hourly increments and create spending alerts. You can also predict future billing based on current usage trends.

Get started: To learn more about using the Billing APIs, see [Azure consumption API overview](#)

Predict future costs

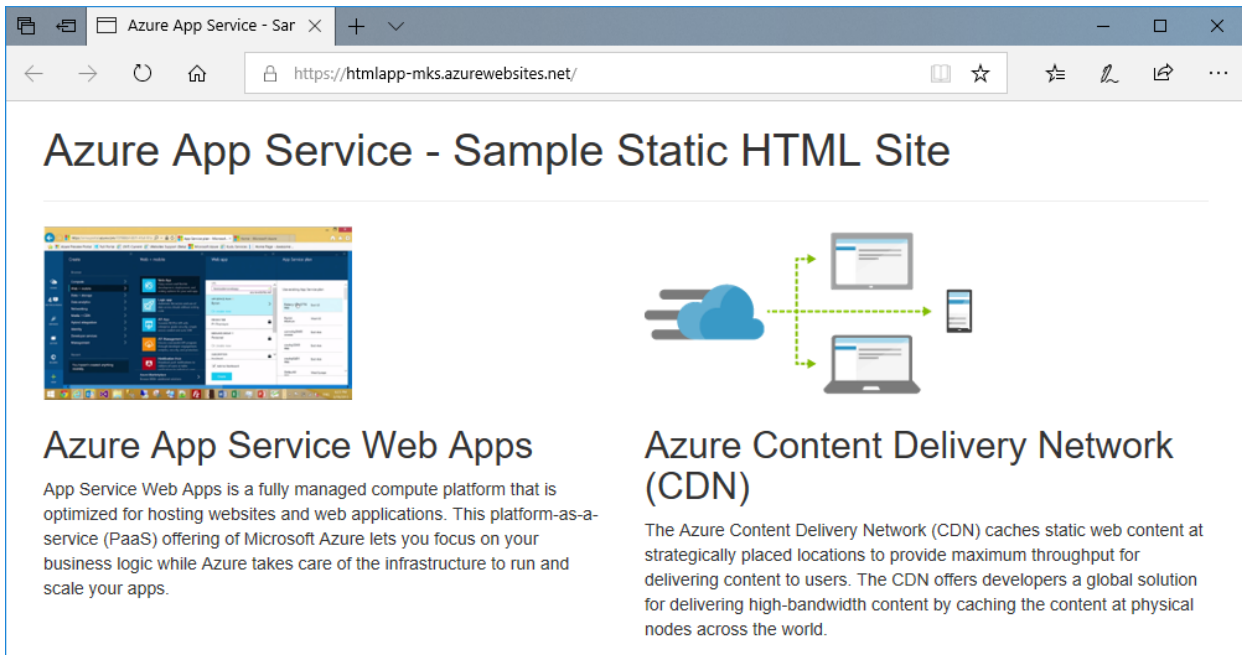
Although it's challenging to estimate costs ahead of time, Azure has tools that can help. It has a [pricing calculator](#) to help estimate the cost of deployed resources. You can also use the Billing resources in the portal and the Billing REST APIs to estimate future costs, based on current consumption.

Get started: See [Azure consumption API overview](#).

Create a static HTML web app in Azure

4/13/2021 • 3 minutes to read • [Edit Online](#)

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



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

Use Azure Cloud Shell

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

To start Azure Cloud Shell:

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

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

1. Start Cloud Shell.

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

Download the sample

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

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

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

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

Create a web app

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

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

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

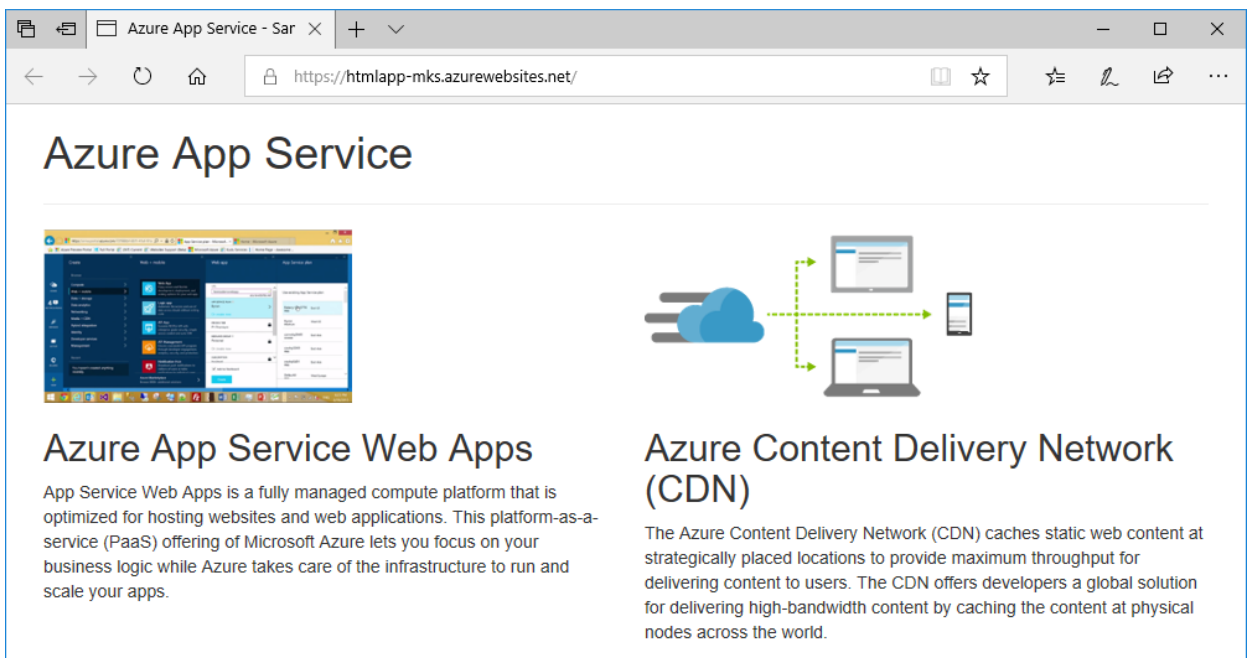
- Create a default resource group.
- Create a default app service plan.
- Create an app with the specified name.
- [Zip deploy](#) files from the current working directory to the web app.

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

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

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

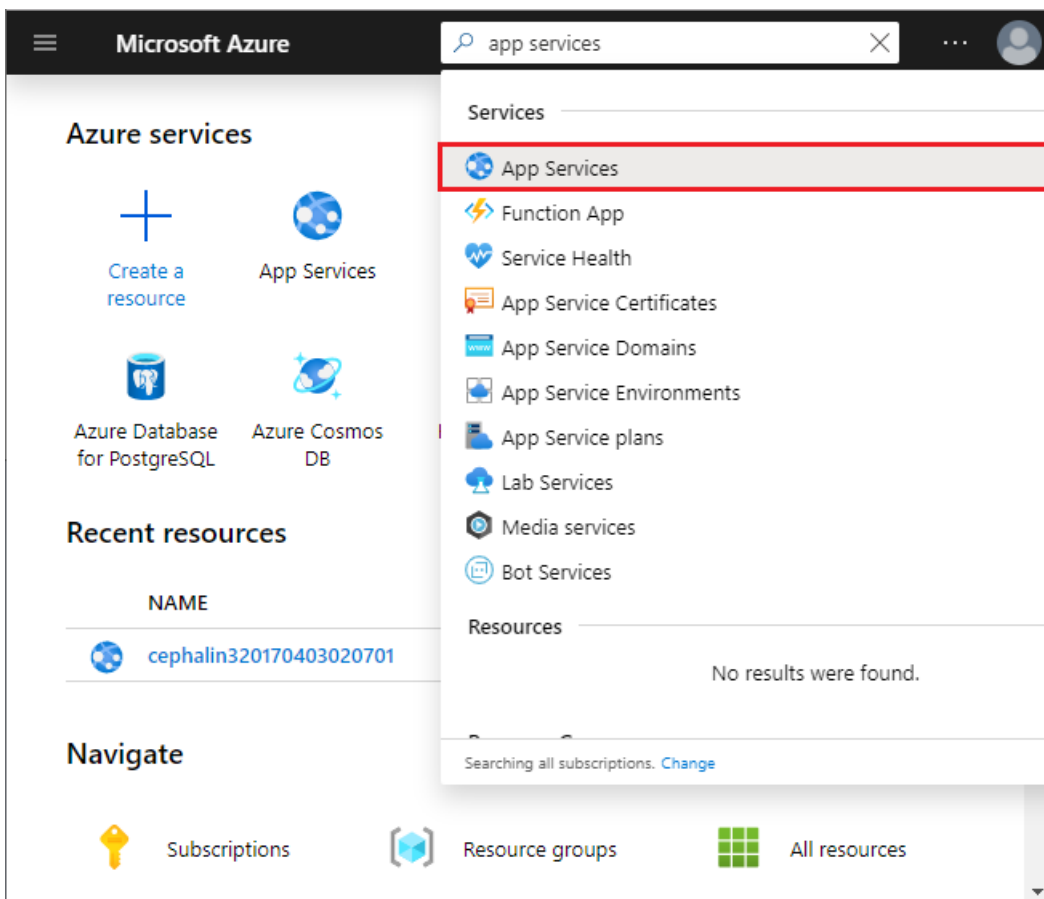
Browse to the app



The screenshot shows a web browser window with the URL <https://htmlapp-mks.azurewebsites.net/>. The page title is "Azure App Service". It features a navigation menu on the left and two main content areas. The left area is titled "Azure App Service Web Apps" and includes a sub-header "Azure App Service Web Apps" and a paragraph: "App Service Web Apps is a fully managed compute platform that is optimized for hosting websites and web applications. This platform-as-a-service (PaaS) offering of Microsoft Azure lets you focus on your business logic while Azure takes care of the infrastructure to run and scale your apps." The right area is titled "Azure Content Delivery Network (CDN)" and includes a sub-header "Azure Content Delivery Network (CDN)" and a paragraph: "The Azure Content Delivery Network (CDN) caches static web content at strategically placed locations to provide maximum throughput for delivering content to users. The CDN offers developers a global solution for delivering high-bandwidth content by caching the content at physical nodes across the world." There is also a diagram showing a cloud icon connected to a laptop and a smartphone.

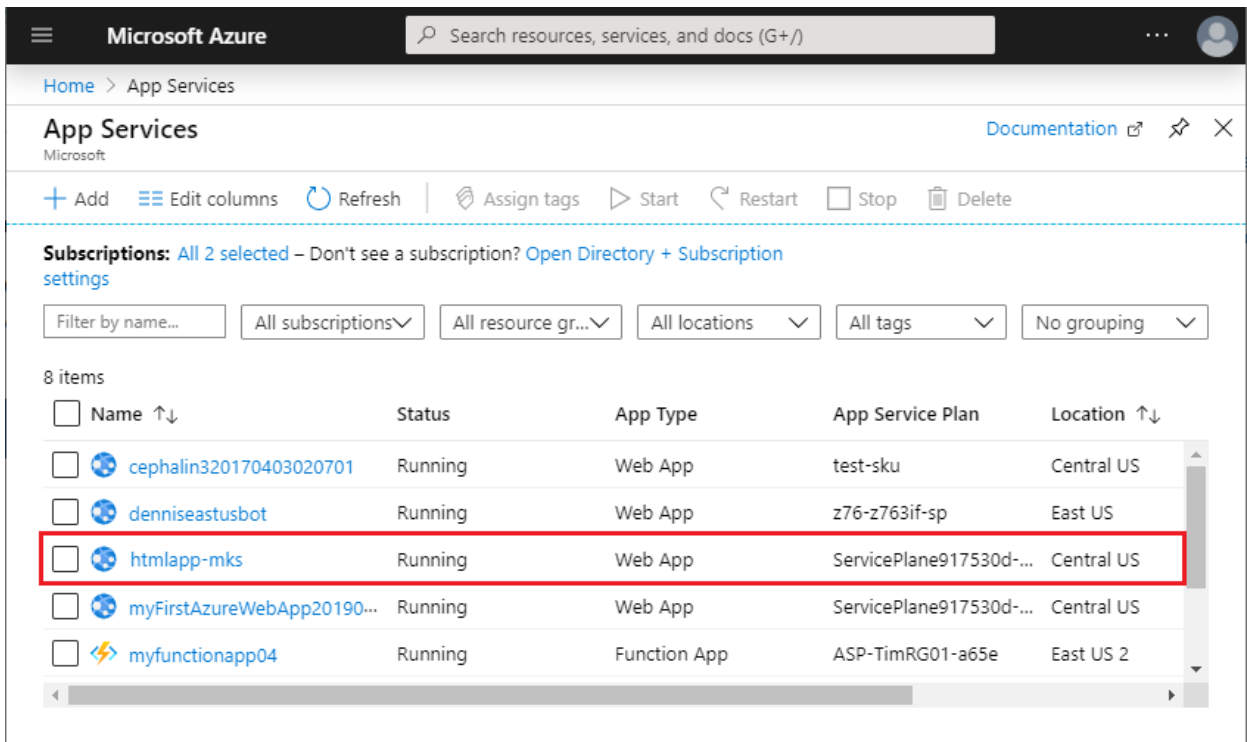
Manage your new Azure app

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

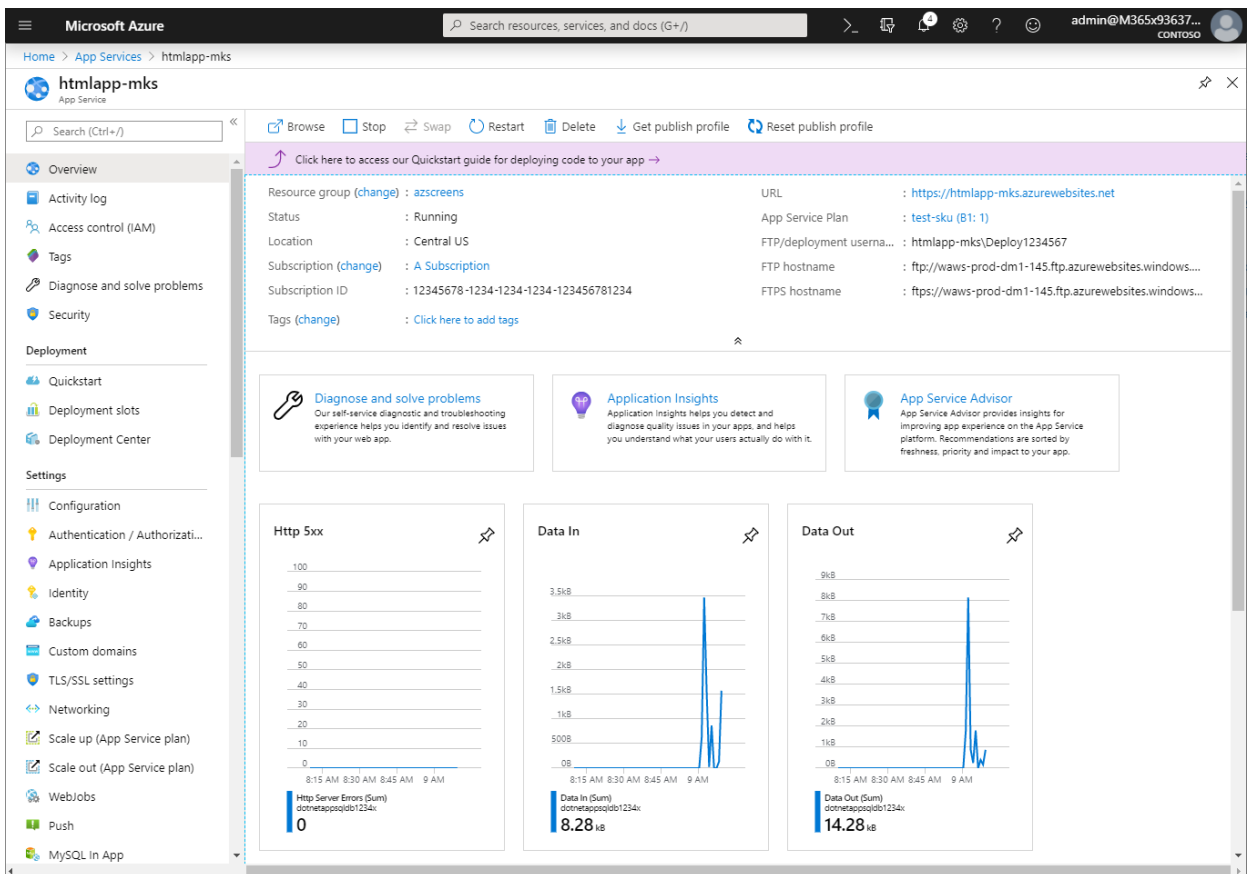


The screenshot shows the Microsoft Azure portal interface. The search bar at the top contains the text "app services". A dropdown menu is open, displaying a list of services. The "App Services" option is highlighted with a red border. Other services listed include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. Below the search results, there is a "Resources" section with the text "No results were found." and a "Searching all subscriptions. Change" link. The main content area on the left shows "Azure services" with icons for "Create a resource" and "App Services", and "Recent resources" with a table containing one resource named "cephalin320170403020701". The "Navigate" section at the bottom includes icons for "Subscriptions", "Resource groups", and "All resources".

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



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, delete the resource group by running the following command in the Cloud Shell. Remember that the resource group name was automatically generated for you in the [create a web app](#) step.

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

This command may take a minute to run.

Next steps

[Map custom domain](#)

Quickstart: Create a Linux virtual machine in the Azure portal

4/14/2021 • 3 minutes to read • [Edit Online](#)

Azure virtual machines (VMs) can be created through the Azure portal. The Azure portal is a browser-based user interface to create Azure resources. This quickstart shows you how to use the Azure portal to deploy a Linux virtual machine (VM) running Ubuntu 18.04 LTS. To see your VM in action, you also SSH to the VM and install the NGINX web server.

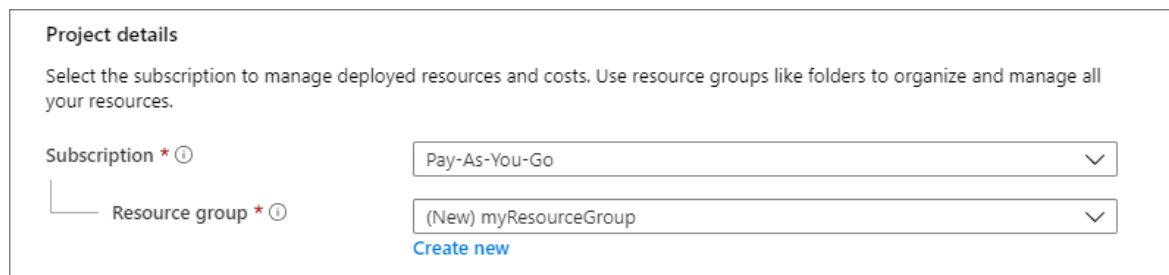
If you don't have an Azure subscription, create a [free account](#) before you begin.

Sign in to Azure

Sign in to the [Azure portal](#) if you haven't already.

Create virtual machine

1. Type **virtual machines** in the search.
2. Under **Services**, select **Virtual machines**.
3. In the **Virtual machines** page, select **Add**. The **Create a virtual machine** page opens.
4. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and then choose to **Create new** resource group. Type *myResourceGroup* for the name.*.



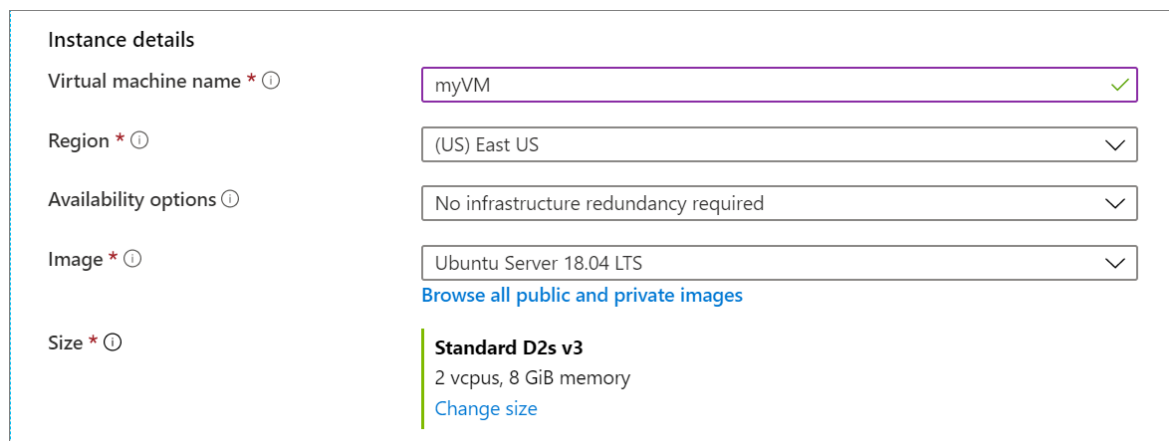
Project details

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

Subscription * ⓘ

Resource group * ⓘ
[Create new](#)

5. Under **Instance details**, type *myVM* for the **Virtual machine name**, choose *East US* for your **Region**, and choose *Ubuntu 18.04 LTS* for your **Image**. Leave the other defaults.



Instance details

Virtual machine name * ⓘ

Region * ⓘ

Availability options ⓘ

Image * ⓘ
[Browse all public and private images](#)

Size * ⓘ **Standard D2s v3**
2 vcpus, 8 GiB memory
[Change size](#)

6. Under **Administrator account**, select **SSH public key**.
7. In **Username** type *azureuser*.

- For SSH public key source, leave the default of **Generate new key pair**, and then type *myKey* for the Key pair name.

Administrator account

Authentication type SSH public key Password

Username * ✓

SSH public key source ▼

Key pair name * ✓

- Under **Inbound port rules > Public inbound ports**, choose **Allow selected ports** and then select **SSH (22)** and **HTTP (80)** from the drop-down.

Inbound port rules


Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * None Allow selected ports

Select inbound ports * ▼

⚠ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

- Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.
- On the **Create a virtual machine** page, you can see the details about the VM you are about to create. When you are ready, select **Create**.
- When the **Generate new key pair** window opens, select **Download private key and create resource**. Your key file will be download as **myKey.pem**. Make sure you know where the `.pem` file was downloaded, you will need the path to it in the next step.
- When the deployment is finished, select **Go to resource**.
- On the page for your new VM, select the public IP address and copy it to your clipboard.

Operating system	: Linux (ubuntu 14.04 LTS)
Size	: Standard D2s v3 (2 vCPUs, 32 GB memory)
Public IP address	: 10.111.12.123 

Copy to clipboard

NOTE

Azure provides an ephemeral IP for Azure Virtual Machines which aren't assigned a public IP address, or are in the backend pool of an internal Basic Azure Load Balancer. The ephemeral IP mechanism provides an outbound IP address that isn't configurable.

The ephemeral IP is disabled when a public IP address is assigned to the virtual machine or the virtual machine is placed in the backend pool of a Standard Load Balancer with or without outbound rules. If a [Azure Virtual Network NAT](#) gateway resource is assigned to the subnet of the virtual machine, the ephemeral IP is disabled.

For more information on outbound connections in Azure, see [Using Source Network Address Translation \(SNAT\) for outbound connections](#).

Connect to virtual machine

Create an SSH connection with the VM.

1. If you are on a Mac or Linux machine, open a Bash prompt. If you are on a Windows machine, open a PowerShell prompt.
2. At your prompt, open an SSH connection to your virtual machine. Replace the IP address with the one from your VM, and replace the path to the `.pem` with the path to where the key file was downloaded.

```
ssh -i .\Downloads\myKey1.pem azureuser@10.111.12.123
```

TIP

The SSH key you created can be used the next time you create a VM in Azure. Just select the **Use a key stored in Azure for SSH public key source** the next time you create a VM. You already have the private key on your computer, so you won't need to download anything.

Install web server

To see your VM in action, install the NGINX web server. From your SSH session, update your package sources and then install the latest NGINX package.

```
sudo apt-get -y update
sudo apt-get -y install nginx
```

When done, type `exit` to leave the SSH session.

View the web server in action

Use a web browser of your choice to view the default NGINX welcome page. Type the public IP address of the VM as the web address. The public IP address can be found on the VM overview page or as part of the SSH connection string you used earlier.



Clean up resources

When no longer needed, you can delete the resource group, virtual machine, and all related resources. To do so, select the resource group for the virtual machine, select **Delete**, then confirm the name of the resource group to delete.

Next steps

In this quickstart, you deployed a simple virtual machine, created a Network Security Group and rule, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

[Azure Linux virtual machine tutorials](#)

Quickstart: Create a Windows virtual machine in the Azure portal

6/14/2021 • 3 minutes to read • [Edit Online](#)

Azure virtual machines (VMs) can be created through the Azure portal. This method provides a browser-based user interface to create VMs and their associated resources. This quickstart shows you how to use the Azure portal to deploy a virtual machine (VM) in Azure that runs Windows Server 2019. To see your VM in action, you then RDP to the VM and install the IIS web server.

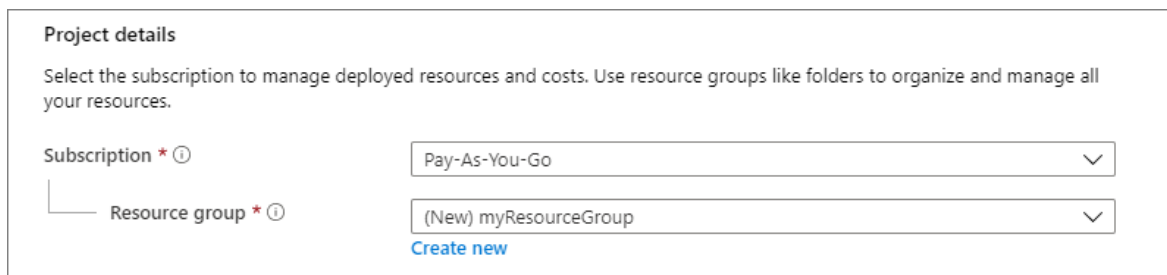
If you don't have an Azure subscription, create a [free account](#) before you begin.

Sign in to Azure

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

Create virtual machine

1. Type **virtual machines** in the search.
2. Under **Services**, select **Virtual machines**.
3. In the **Virtual machines** page, select **Add** then **Virtual machine**.
4. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and then choose to **Create new** resource group. Type *myResourceGroup* for the name.



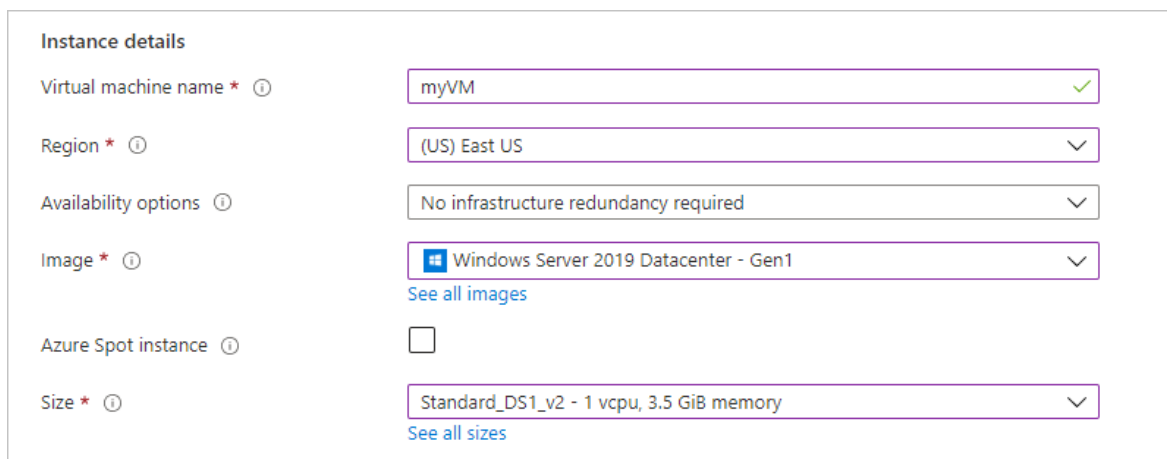
Project details

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

Subscription * ⓘ

Resource group * ⓘ [Create new](#)

5. Under **Instance details**, type *myVM* for the **Virtual machine name** and choose *East US* for your **Region**. Choose *Windows Server 2019 Datacenter* for the **Image** and *Standard_DS1_v2* for the **Size**. Leave the other defaults.



Instance details

Virtual machine name * ⓘ

Region * ⓘ

Availability options ⓘ

Image * ⓘ [See all images](#)

Azure Spot instance ⓘ

Size * ⓘ [See all sizes](#)

6. Under **Administrator account**, provide a username, such as *azureuser* and a password. The password

must be at least 12 characters long and meet the [defined complexity requirements](#).

Administrator account	
Username * ⓘ	<input type="text" value="azureuser"/> ✓
Password * ⓘ	<input type="password" value="....."/> ✓
Confirm password * ⓘ	<input type="password" value="....."/> ✓

- Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP (80)** from the drop-down.

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * ⓘ None Allow selected ports

Select inbound ports * ✓

⚠ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

- Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.

Licensing

Save up to 49% with a license you already own using Azure Hybrid Benefit. [Learn more](#) ↗

Would you like to use an existing Windows Server license? * ⓘ

[Review Azure hybrid benefit compliance](#)

Review + create < Previous Next : Disks >

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

- After deployment is complete, select **Go to resource**.

^ **Next steps**

[Setup auto-shutdown](#) Recommended

[Monitor VM health, performance and network dependencies](#) Recommended

[Run a script inside the virtual machine](#) Recommended

Go to resource Create another VM

NOTE

Azure provides an ephemeral IP for Azure Virtual Machines which aren't assigned a public IP address, or are in the backend pool of an internal Basic Azure Load Balancer. The ephemeral IP mechanism provides an outbound IP address that isn't configurable.

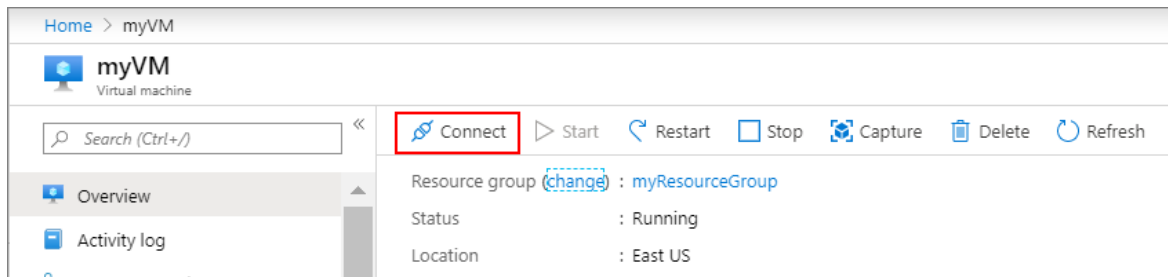
The ephemeral IP is disabled when a public IP address is assigned to the virtual machine or the virtual machine is placed in the backend pool of a Standard Load Balancer with or without outbound rules. If a [Azure Virtual Network NAT](#) gateway resource is assigned to the subnet of the virtual machine, the ephemeral IP is disabled.

For more information on outbound connections in Azure, see [Using Source Network Address Translation \(SNAT\) for outbound connections](#).

Connect to virtual machine

Create a remote desktop connection to the virtual machine. These directions tell you how to connect to your VM from a Windows computer. On a Mac, you need an RDP client such as this [Remote Desktop Client](#) from the Mac App Store.

1. On the overview page for your virtual machine, select the **Connect** button then select **RDP**.



2. In the **Connect with RDP** page, keep the default options to connect by IP address, over port 3389, and click **Download RDP file**.
3. Open the downloaded RDP file and click **Connect** when prompted.
4. In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username as `localhost\username`, enter the password you created for the virtual machine, and then click **OK**.
5. You may receive a certificate warning during the sign-in process. Click **Yes** or **Continue** to create the connection.

Install web server

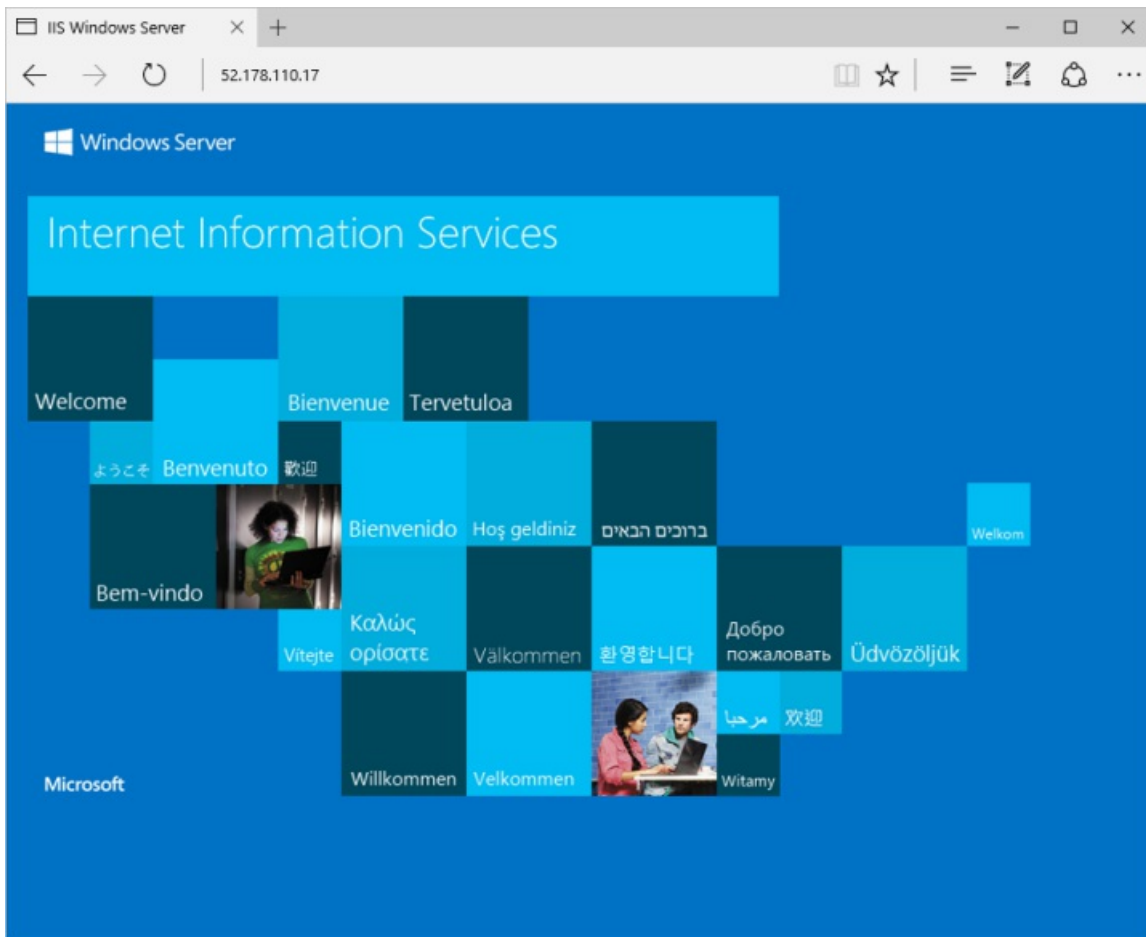
To see your VM in action, install the IIS web server. Open a PowerShell prompt on the VM and run the following command:

```
Install-WindowsFeature -name Web-Server -IncludeManagementTools
```

When done, close the RDP connection to the VM.

View the IIS welcome page

In the portal, select the VM and in the overview of the VM, hover over the IP address to show **Copy to clipboard**. Copy the IP address and paste it into a browser tab. The default IIS welcome page will open, and should look like this:



Clean up resources

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

Go to the resource group for the virtual machine, then select **Delete resource group**. Confirm the name of the resource group to finish deleting the resources.

Next steps

In this quickstart, you deployed a simple virtual machine, opened a network port for web traffic, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Windows VMs.

[Azure Windows virtual machine tutorials](#)

Getting started with Azure Functions

3/5/2021 • 3 minutes to read • [Edit Online](#)

Introduction

[Azure Functions](#) allows you to implement your system's logic into readily-available blocks of code. These code blocks are called "functions".

Use the following resources to get started.

ACTION	RESOURCES
Create your first function	Using one of the following tools: <ul style="list-style-type: none">• Visual Studio• Visual Studio Code• Command line
See a function running	<ul style="list-style-type: none">• Azure Samples Browser• Azure Community Library
Explore an interactive tutorial	<ul style="list-style-type: none">• Choose the best Azure serverless technology for your business scenario• Well-Architected Framework - Performance efficiency• Execute an Azure Function with triggers <p>See Microsoft Learn for a full listing of interactive tutorials.</p>
Review best practices	<ul style="list-style-type: none">• Performance and reliability• Manage connections• Error handling and function retries• Security
Learn more in-depth	<ul style="list-style-type: none">• Learn how functions automatically increase or decrease instances to match demand• Explore the different deployment methods available• Use built-in monitoring tools to help analyze your functions• Read the C# language reference
ACTION	RESOURCES
Create your first function	Using one of the following tools: <ul style="list-style-type: none">• Visual Studio Code• Java/Maven function with terminal/command prompt• Gradle• Eclipse• IntelliJ IDEA
See a function running	<ul style="list-style-type: none">• Azure Samples Browser• Azure Community Library

ACTION	RESOURCES
Explore an interactive tutorial	<ul style="list-style-type: none"> • Choose the best Azure serverless technology for your business scenario • Well-Architected Framework - Performance efficiency • Develop an App using the Maven Plugin for Azure Functions <p>See Microsoft Learn for a full listing of interactive tutorials.</p>
Review best practices	<ul style="list-style-type: none"> • Performance and reliability • Manage connections • Error handling and function retries • Security
Learn more in-depth	<ul style="list-style-type: none"> • Learn how functions automatically increase or decrease instances to match demand • Explore the different deployment methods available • Use built-in monitoring tools to help analyze your functions • Read the Java language reference

ACTION	RESOURCES
Create your first function	<p>Using one of the following tools:</p> <ul style="list-style-type: none"> • Visual Studio Code • Node.js terminal/command prompt
See a function running	<ul style="list-style-type: none"> • Azure Samples Browser • Azure Community Library
Explore an interactive tutorial	<ul style="list-style-type: none"> • Choose the best Azure serverless technology for your business scenario • Well-Architected Framework - Performance efficiency • Build Serverless APIs with Azure Functions • Create serverless logic with Azure Functions • Refactor Node.js and Express APIs to Serverless APIs with Azure Functions <p>See Microsoft Learn for a full listing of interactive tutorials.</p>
Review best practices	<ul style="list-style-type: none"> • Performance and reliability • Manage connections • Error handling and function retries • Security
Learn more in-depth	<ul style="list-style-type: none"> • Learn how functions automatically increase or decrease instances to match demand • Explore the different deployment methods available • Use built-in monitoring tools to help analyze your functions • Read the JavaScript or TypeScript language reference

ACTION	RESOURCES
Create your first function	<ul style="list-style-type: none"> • Using Visual Studio Code

ACTION	RESOURCES
See a function running	<ul style="list-style-type: none"> • Azure Samples Browser • Azure Community Library
Explore an interactive tutorial	<ul style="list-style-type: none"> • Choose the best Azure serverless technology for your business scenario • Well-Architected Framework - Performance efficiency • Build Serverless APIs with Azure Functions • Create serverless logic with Azure Functions • Execute an Azure Function with triggers <p>See Microsoft Learn for a full listing of interactive tutorials.</p>
Review best practices	<ul style="list-style-type: none"> • Performance and reliability • Manage connections • Error handling and function retries • Security
Learn more in-depth	<ul style="list-style-type: none"> • Learn how functions automatically increase or decrease instances to match demand • Explore the different deployment methods available • Use built-in monitoring tools to help analyze your functions • Read the PowerShell language reference)

ACTION	RESOURCES
Create your first function	<p>Using one of the following tools:</p> <ul style="list-style-type: none"> • Visual Studio Code • Terminal/command prompt
See a function running	<ul style="list-style-type: none"> • Azure Samples Browser • Azure Community Library
Explore an interactive tutorial	<ul style="list-style-type: none"> • Choose the best Azure serverless technology for your business scenario • Well-Architected Framework - Performance efficiency • Build Serverless APIs with Azure Functions • Create serverless logic with Azure Functions <p>See Microsoft Learn for a full listing of interactive tutorials.</p>
Review best practices	<ul style="list-style-type: none"> • Performance and reliability • Manage connections • Error handling and function retries • Security • Improve throughput performance
Learn more in-depth	<ul style="list-style-type: none"> • Learn how functions automatically increase or decrease instances to match demand • Explore the different deployment methods available • Use built-in monitoring tools to help analyze your functions • Read the Python language reference

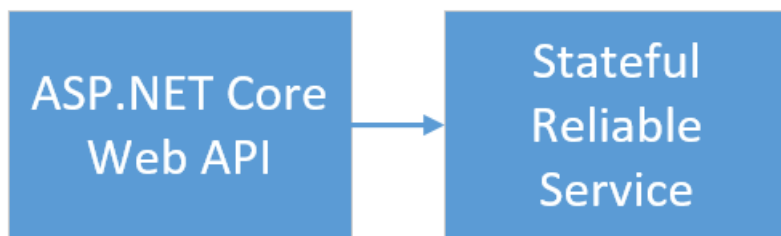
Next steps

[Learn about the anatomy of an Azure Functions application](#)

Tutorial: Create and deploy an application with an ASP.NET Core Web API front-end service and a stateful back-end service

3/5/2021 • 14 minutes to read • [Edit Online](#)

This tutorial is part one of a series. You will learn how to create an Azure Service Fabric application with an ASP.NET Core Web API front end and a stateful back-end service to store your data. When you're finished, you have a voting application with an ASP.NET Core web front-end that saves voting results in a stateful back-end service in the cluster. This tutorial series requires a Windows developer machine. If you don't want to manually create the voting application, you can [download the source code](#) for the completed application and skip ahead to [Walk through the voting sample application](#). If you prefer, you can also watch a [video walk-through](#) of this tutorial.



In part one of the series, you learn how to:

- Create an ASP.NET Core Web API service as a stateful reliable service
- Create an ASP.NET Core Web Application service as a stateless web service
- Use the reverse proxy to communicate with the stateful service

In this tutorial series you learn how to:

- Build a .NET Service Fabric application
- [Deploy the application to a remote cluster](#)
- [Add an HTTPS endpoint to an ASP.NET Core front-end service](#)
- [Configure CI/CD using Azure Pipelines](#)
- [Set up monitoring and diagnostics for the application](#)

Prerequisites

Before you begin this tutorial:

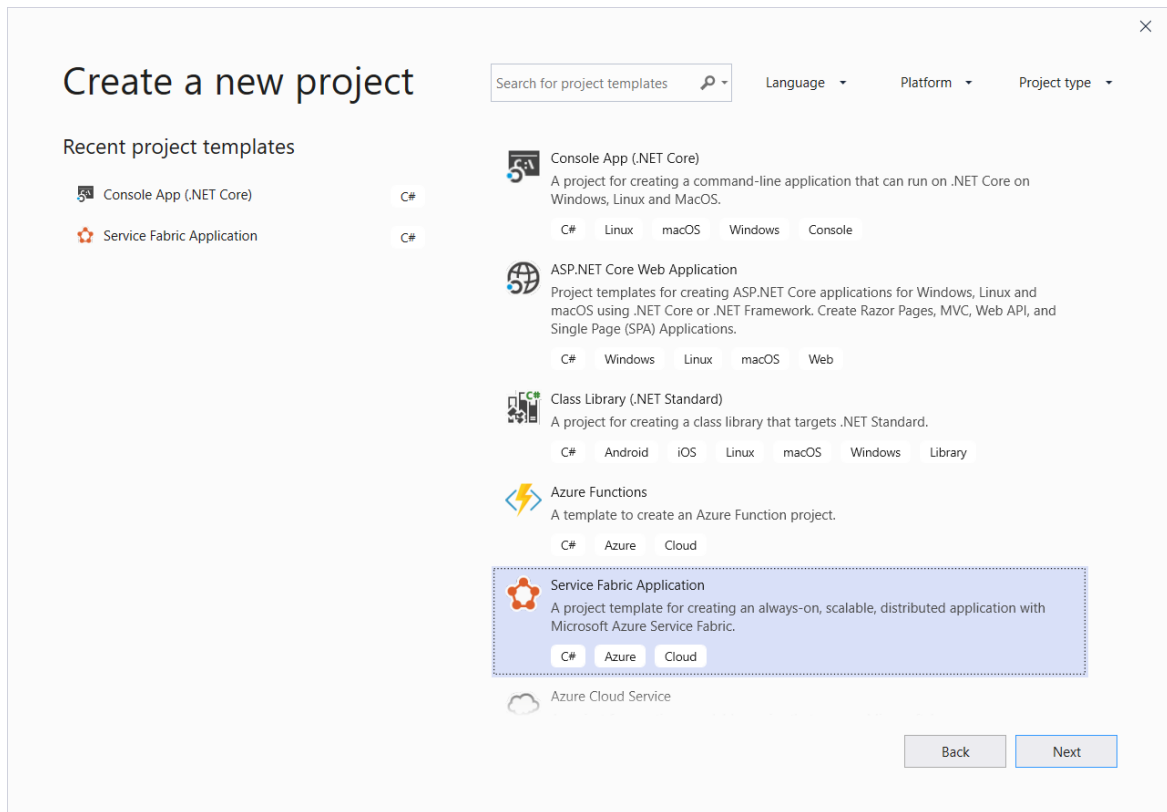
- If you don't have an Azure subscription, create a [free account](#)
- [Install Visual Studio 2019](#) version 15.5 or later with the **Azure development** and **ASP.NET and web development** workloads.
- [Install the Service Fabric SDK](#)

Create an ASP.NET Web API service as a reliable service

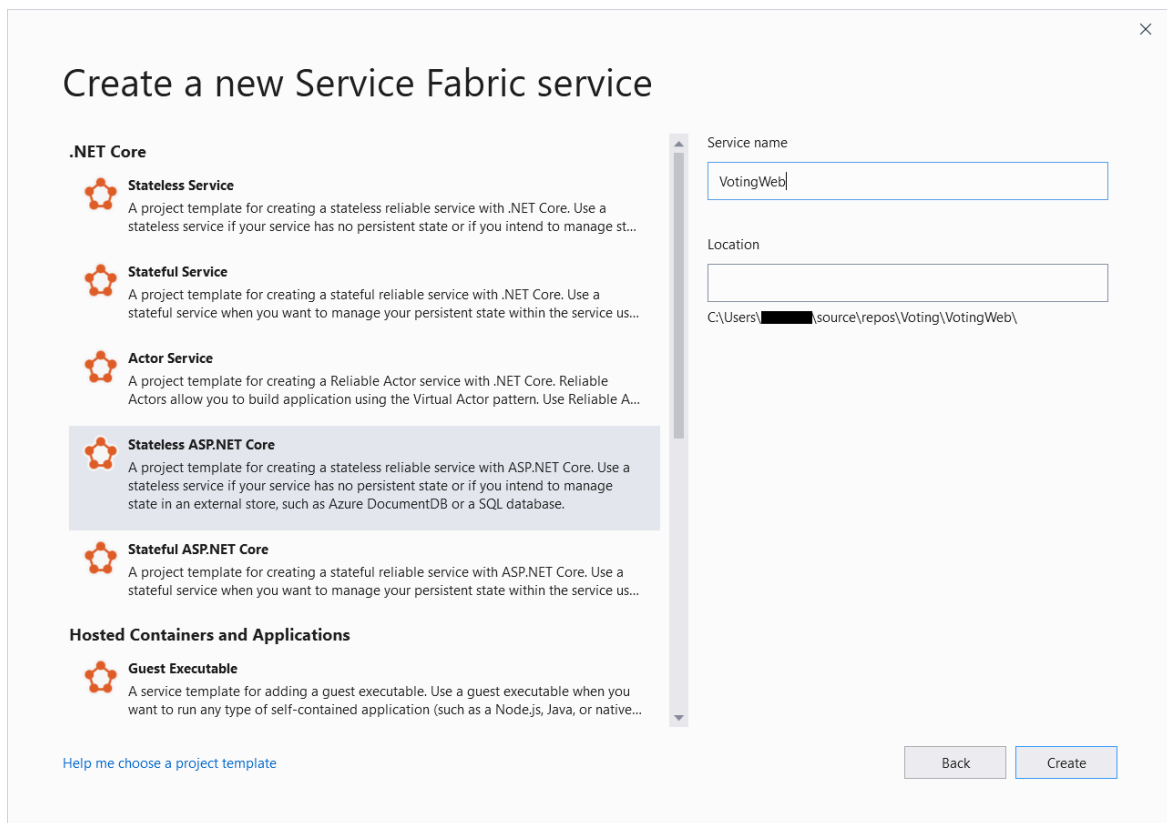
First, create the web front-end of the voting application using ASP.NET Core. ASP.NET Core is a lightweight, cross-platform web development framework that you can use to create modern web UI and web APIs. To get a complete understanding of how ASP.NET Core integrates with Service Fabric, we strongly recommend reading

through the [ASP.NET Core in Service Fabric Reliable Services](#) article. For now, you can follow this tutorial to get started quickly. To learn more about ASP.NET Core, see the [ASP.NET Core Documentation](#).

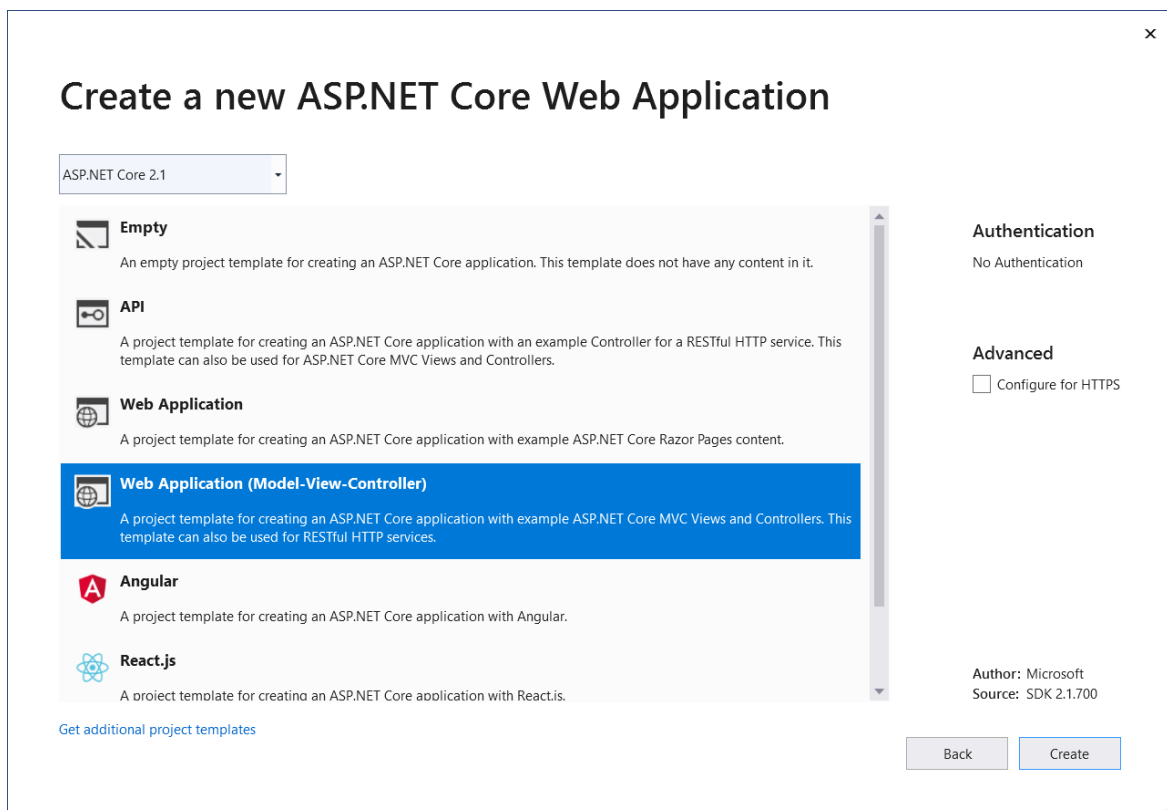
1. Launch Visual Studio as an **administrator**.
2. Create a project with **File->New->Project**.
3. In the **New Project** dialog, choose **Cloud > Service Fabric Application**.
4. Name the application **Voting** and click **OK**.



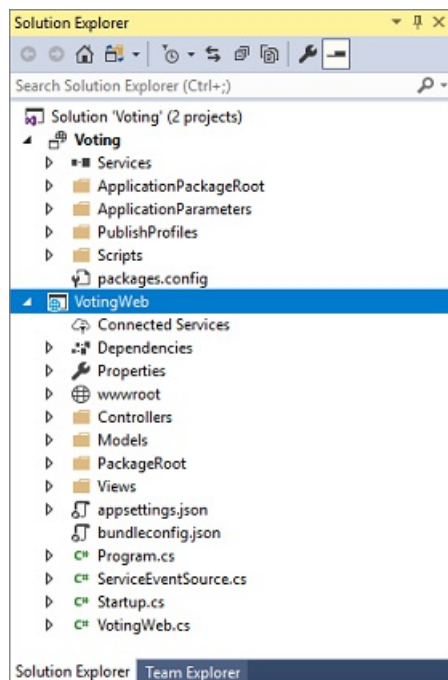
5. On the **New Service Fabric Service** page, choose **Stateless ASP.NET Core**, name your service **VotingWeb**, then click **OK**.



- The next page provides a set of ASP.NET Core project templates. For this tutorial, choose **Web Application (Model-View-Controller)**, then click OK.



Visual Studio creates an application and a service project and displays them in Solution Explorer.



Update the site.js file

Open `wwwroot/js/site.js`. Replace its contents with the following JavaScript used by the Home views, then save your changes.

```
var app = angular.module('VotingApp', ['ui.bootstrap']);
app.run(function () { });

app.controller('VotingAppController', ['$rootScope', '$scope', '$http', '$timeout', function ($rootScope,
$scope, $http, $timeout) {

    $scope.refresh = function () {
        $http.get('api/Votes?c=' + new Date().getTime())
            .then(function (data, status) {
                $scope.votes = data;
            }, function (data, status) {
                $scope.votes = undefined;
            });
    };

    $scope.remove = function (item) {
        $http.delete('api/Votes/' + item)
            .then(function (data, status) {
                $scope.refresh();
            })
    };

    $scope.add = function (item) {
        var fd = new FormData();
        fd.append('item', item);
        $http.put('api/Votes/' + item, fd, {
            transformRequest: angular.identity,
            headers: { 'Content-Type': undefined }
        })
            .then(function (data, status) {
                $scope.refresh();
                $scope.item = undefined;
            })
    };
}]);
```

Update the Index.cshtml file

Open **Views/Home/Index.cshtml**, the view specific to the Home controller. Replace its contents with the following, then save your changes.

```
@{
    ViewData["Title"] = "Service Fabric Voting Sample";
}

<div ng-controller="VotingAppController" ng-init="refresh()">
    <div class="container-fluid">
        <div class="row">
            <div class="col-xs-8 col-xs-offset-2 text-center">
                <h2>Service Fabric Voting Sample</h2>
            </div>
        </div>

        <div class="row">
            <div class="col-xs-8 col-xs-offset-2">
                <form class="col-xs-12 center-block">
                    <div class="col-xs-6 form-group">
                        <input id="txtAdd" type="text" class="form-control" placeholder="Add voting option"
ng-model="item"/>
                    </div>
                    <button id="btnAdd" class="btn btn-default" ng-click="add(item)">
                        <span class="glyphicon glyphicon-plus" aria-hidden="true"></span>
                        Add
                    </button>
                </form>
            </div>
        </div>

        <hr/>

        <div class="row">
            <div class="col-xs-8 col-xs-offset-2">
                <div class="row">
                    <div class="col-xs-4">
                        Click to vote
                    </div>
                </div>
                <div class="row top-buffer" ng-repeat="vote in votes.data">
                    <div class="col-xs-8">
                        <button class="btn btn-success text-left btn-block" ng-click="add(vote.key)">
                            <span class="pull-left">
                                {{vote.Key}}
                            </span>
                            <span class="badge pull-right">
                                {{vote.Value}} Votes
                            </span>
                        </button>
                    </div>
                    <div class="col-xs-4">
                        <button class="btn btn-danger pull-right btn-block" ng-click="remove(vote.key)">
                            <span class="glyphicon glyphicon-remove" aria-hidden="true"></span>
                            Remove
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Update the **_Layout.cshtml** file

Open **Views/Shared/_Layout.cshtml**, the default layout for the ASP.NET app. Replace its contents with the following, then save your changes.

```

<!DOCTYPE html>
<html ng-app="VotingApp" xmlns:ng="https://angularjs.org">
<head>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>@ViewData["Title"]</title>

  <link href="~/lib/bootstrap/dist/css/bootstrap.css" rel="stylesheet"/>
  <link href="~/css/site.css" rel="stylesheet"/>

</head>
<body>
<div class="container body-content">
  @RenderBody()
</div>

<script src="~/lib/jquery/dist/jquery.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.7.2/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/2.5.0/ui-bootstrap-tpls.js">
</script>
<script src="~/js/site.js"></script>

@RenderSection("Scripts", required: false)
</body>
</html>

```

Update the VotingWeb.cs file

Open the *VotingWeb.cs* file, which creates the ASP.NET Core WebHost inside the stateless service using the WebListener web server.

Add the `using System.Net.Http;` directive to the top of the file.

Replace the `CreateServiceInstanceListeners()` function with the following code, then save your changes.

```

protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
{
    return new ServiceInstanceListener[]
    {
        new ServiceInstanceListener(
            serviceContext =>
                new KestrelCommunicationListener(
                    serviceContext,
                    "ServiceEndpoint",
                    (url, listener) =>
                    {
                        ServiceEventSource.Current.ServiceMessage(serviceContext, $"Starting Kestrel on
{url}");

                        return new WebHostBuilder()
                            .UseKestrel()
                            .ConfigureServices(
                                services => services
                                    .AddSingleton<HttpClient>(new HttpClient())
                                    .AddSingleton<FabricClient>(new FabricClient())
                                    .AddSingleton<StatelessServiceContext>(serviceContext))
                            .UseContentRoot(Directory.GetCurrentDirectory())
                            .UseStartup<Startup>()
                            .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.None)
                            .UseUrls(url)
                            .Build();
                    }
                )))
    };
}

```

Also add the following `GetVotingDataServiceName` method below `CreateServiceInstanceListeners()`, then save your changes. `GetVotingDataServiceName` returns the service name when polled.

```

internal static Uri GetVotingDataServiceName(ServiceContext context)
{
    return new Uri($"{context.CodePackageActivationContext.ApplicationName}/VotingData");
}

```

Add the `VotesController.cs` file

Add a controller, which defines voting actions. Right-click on the **Controllers** folder, then select **Add->New item->Visual C#->ASP.NET Core->Class**. Name the file `VotesController.cs`, then click **Add**.

Replace the `VotesController.cs` file contents with the following, then save your changes. Later, in [Update the VotesController.cs file](#), this file is modified to read and write voting data from the back-end service. For now, the controller returns static string data to the view.

```

namespace VotingWeb.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Fabric;
    using System.Fabric.Query;
    using System.Linq;
    using System.Net.Http;
    using System.Net.Http.Headers;
    using System.Text;
    using System.Threading.Tasks;
    using Microsoft.AspNetCore.Mvc;
    using Newtonsoft.Json;

    [Produces("application/json")]
    [Route("api/Votes")]
    public class VotesController : Controller
    {
        private readonly HttpClient httpClient;

        public VotesController(HttpClient httpClient)
        {
            this.httpClient = httpClient;
        }

        // GET: api/Votes
        [HttpGet]
        public async Task<IActionResult> Get()
        {
            List<KeyValuePair<string, int>> votes= new List<KeyValuePair<string, int>>();
            votes.Add(new KeyValuePair<string, int>("Pizza", 3));
            votes.Add(new KeyValuePair<string, int>("Ice cream", 4));

            return Json(votes);
        }
    }
}

```

Configure the listening port

When the VotingWeb front-end service is created, Visual Studio randomly selects a port for the service to listen on. The VotingWeb service acts as the front-end for this application and accepts external traffic, so let's bind that service to a fixed and well-know port. The [service manifest](#) declares the service endpoints.

In Solution Explorer, open *VotingWeb/PackageRoot/ServiceManifest.xml*. Find the **Endpoint** element in the **Resources** section and change the **Port** value to **8080**. To deploy and run the application locally, the application listening port must be open and available on your computer.

```

<Resources>
  <Endpoints>
    <!-- This endpoint is used by the communication listener to obtain the port on which to
         listen. Please note that if your service is partitioned, this port is shared with
         replicas of different partitions that are placed in your code. -->
    <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" Port="8080" />
  </Endpoints>
</Resources>

```

Also update the Application URL property value in the Voting project so a web browser opens to the correct port when you debug your application. In Solution Explorer, select the **Voting** project and update the **Application URL** property to **8080**.

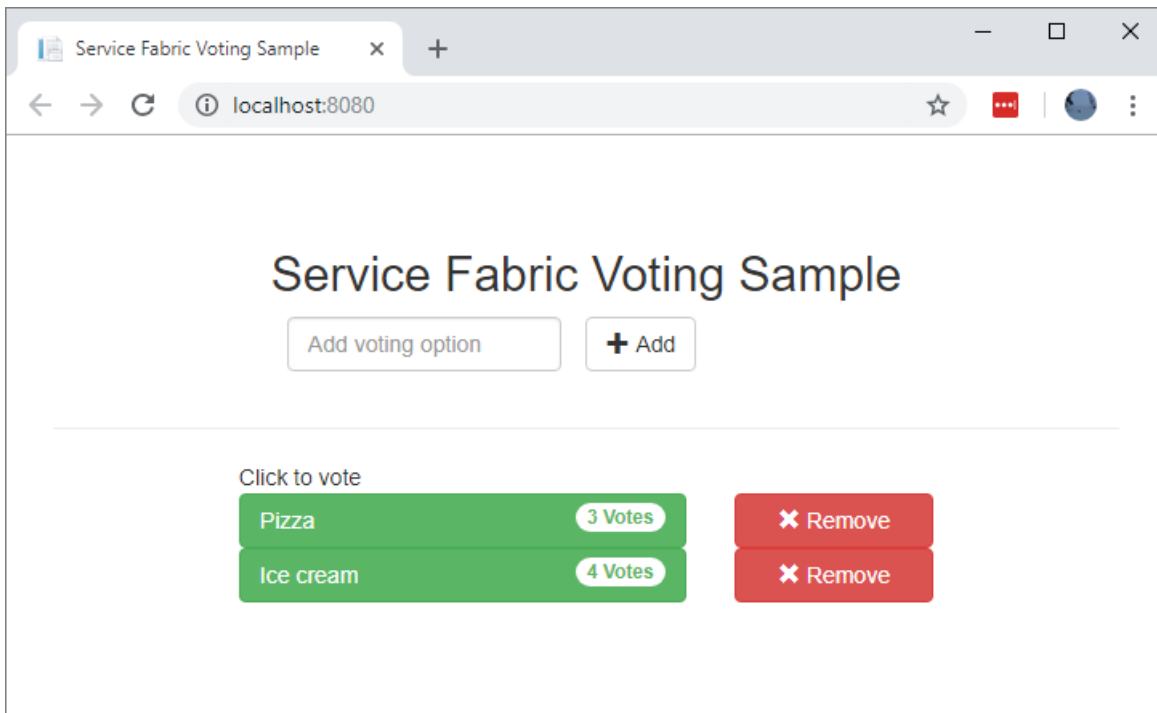
Deploy and run the Voting application locally

You can now go ahead and run the Voting application for debugging. In Visual Studio, press F5 to deploy the application to your local Service Fabric cluster in debug mode. The application will fail if you didn't previously open Visual Studio as **administrator**.

NOTE

The first time you run and deploy the application locally, Visual Studio creates a local Service Fabric cluster for debugging. Cluster creation may take some time. The cluster creation status is displayed in the Visual Studio output window.

After the Voting application has been deployed to your local Service Fabric cluster, your web app will open in a browser tab automatically and should look like this:



To stop debugging the application, go back to Visual Studio and press **Shift+F5**.

Add a stateful back-end service to your application

Now that an ASPNET Web API service is running in the application, go ahead and add a stateful reliable service to store some data in the application.

Service Fabric allows you to consistently and reliably store your data right inside your service by using reliable collections. Reliable collections are a set of highly available and reliable collection classes that are familiar to anyone who has used C# collections.

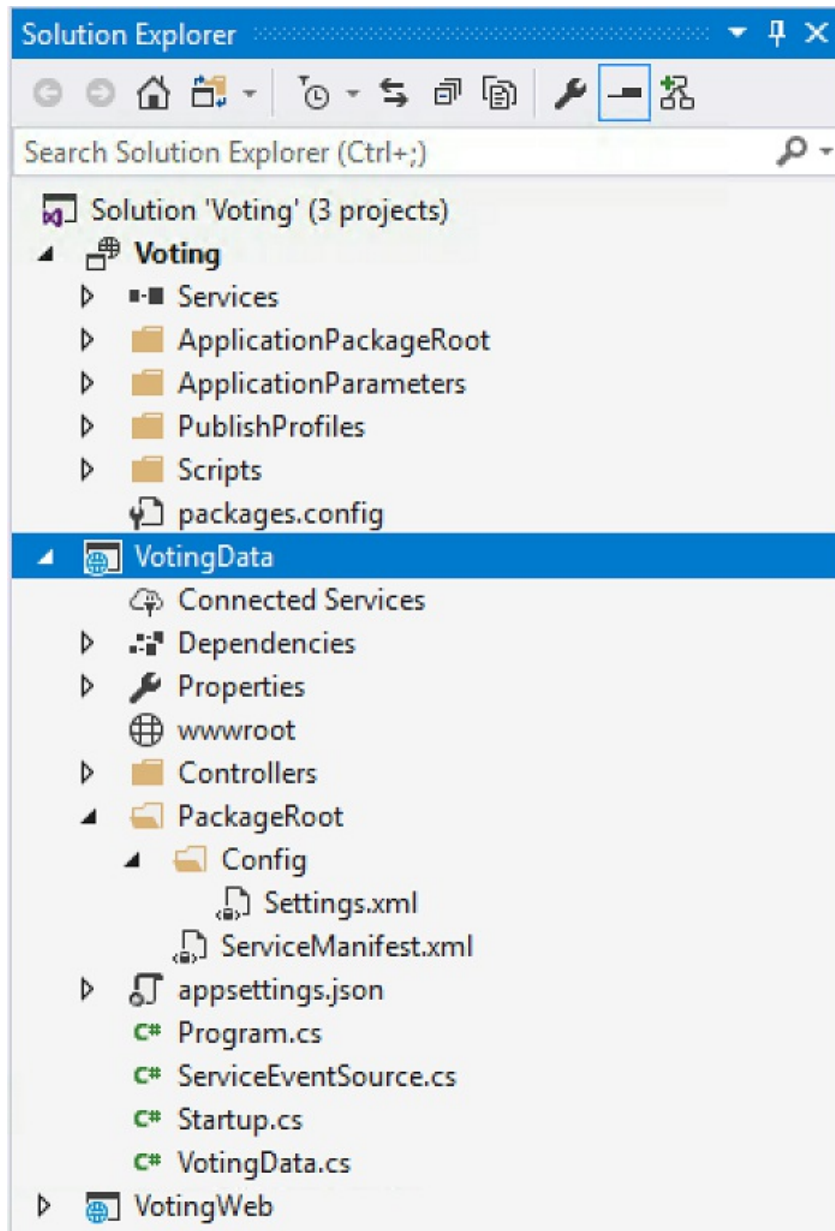
In this tutorial, you create a service which stores a counter value in a reliable collection.

1. In Solution Explorer, right-click **Services** within the Voting application project and choose **Add -> New Service Fabric Service....**
2. In the **New Service Fabric Service** dialog, choose **Stateful ASP.NET Core**, name the service **VotingData**, then press **OK**.

Once your service project is created, you have two services in your application. As you continue to build your application, you can add more services in the same way. Each can be independently versioned and upgraded.

3. The next page provides a set of ASPNET Core project templates. For this tutorial, choose **API**.

Visual Studio creates the VotingData service project and displays it in Solution Explorer.



Add the VoteDataController.cs file

In the **VotingData** project, right-click on the **Controllers** folder, then select **Add->New item->Class**. Name the file **VoteDataController.cs** and click **Add**. Replace the file contents with the following, then save your changes.

```
namespace VotingData.Controllers
{
    using System.Collections.Generic;
    using System.Threading;
    using System.Threading.Tasks;
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.ServiceFabric.Data;
    using Microsoft.ServiceFabric.Data.Collections;

    [Route("api/[controller]")]
    public class VoteDataController : Controller
    {
        private readonly IReliableStateManager stateManager;

        public VoteDataController(IReliableStateManager stateManager)
        {
            this.stateManager = stateManager;
        }
    }
}
```

```

// GET api/VoteData
[HttpGet]
public async Task<IActionResult> Get()
{
    CancellationToken ct = new CancellationToken();

    IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        Microsoft.ServiceFabric.Data.IAsyncEnumerable<KeyValuePair<string, int>> list = await
votesDictionary.CreateEnumerableAsync(tx);

        Microsoft.ServiceFabric.Data.IAsyncEnumerator<KeyValuePair<string, int>> enumerator =
list.GetAsyncEnumerator();

        List<KeyValuePair<string, int>> result = new List<KeyValuePair<string, int>>();

        while (await enumerator.MoveNextAsync(ct))
        {
            result.Add(enumerator.Current);
        }

        return this.Json(result);
    }
}

// PUT api/VoteData/name
[HttpPut("{name}")]
public async Task<IActionResult> Put(string name)
{
    IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
        await tx.CommitAsync();
    }

    return new OkResult();
}

// DELETE api/VoteData/name
[HttpDelete("{name}")]
public async Task<IActionResult> Delete(string name)
{
    IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        if (await votesDictionary.ContainsKeyAsync(tx, name))
        {
            await votesDictionary.TryRemoveAsync(tx, name);
            await tx.CommitAsync();
            return new OkResult();
        }
        else
        {
            return new NotFoundResult();
        }
    }
}
}
}

```

Connect the services

In this next step, connect the two services and make the front-end Web application get and set voting information from the back-end service.

Service Fabric provides complete flexibility in how you communicate with reliable services. Within a single application, you might have services that are accessible via TCP. Other services that might be accessible via an HTTP REST API and still other services could be accessible via web sockets. For background on the options available and the tradeoffs involved, see [Communicating with services](#).

This tutorial uses [ASP.NET Core Web API](#) and the [Service Fabric reverse proxy](#) so the VotingWeb front-end web service can communicate with the back-end VotingData service. The reverse proxy is configured by default to use port 19081 and should work for this tutorial. The reverse proxy port is set in the Azure Resource Manager template used to set up the cluster. To find which port is used, look in the cluster template in the [Microsoft.ServiceFabric/clusters](#) resource:

```
"nodeTypes": [
  {
    ...
    "httpGatewayEndpointPort": "[variables('nt0fabricHttpGatewayPort')]",
    "isPrimary": true,
    "vmInstanceCount": "[parameters('nt0InstanceCount')]",
    "reverseProxyEndpointPort": "[parameters('SFReverseProxyPort')]"
  }
],
```

To find the reverse proxy port used in your local development cluster, view the **HttpApplicationGatewayEndpoint** element in the local Service Fabric cluster manifest:

1. Open a browser window and navigate to <http://localhost:19080> to open the Service Fabric Explorer tool.
2. Select **Cluster** -> **Manifest**.
3. Make a note of the HttpApplicationGatewayEndpoint element port. By default this should be 19081. If it is not 19081, you will need to change the port in the GetProxyAddress method of the following VotesController.cs code.

Update the VotesController.cs file

In the **VotingWeb** project, open the *Controllers/VotesController.cs* file. Replace the `VotesController` class definition contents with the following, then save your changes. If the reverse proxy port you discovered in the pervious step is not 19081, change the port used in the GetProxyAddress method from 19081 to the port that you discovered.

```
public class VotesController : Controller
{
    private readonly HttpClient httpClient;
    private readonly FabricClient fabricClient;
    private readonly StatelessServiceContext serviceContext;

    public VotesController(HttpClient httpClient, StatelessServiceContext context, FabricClient fabricClient)
    {
        this.fabricClient = fabricClient;
        this.httpClient = httpClient;
        this.serviceContext = context;
    }

    // GET: api/Votes
    [HttpGet("")]
```

```

public async Task<IActionResult> Get()
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);

    ServicePartitionList partitions = await
this.fabricClient.QueryManager.GetPartitionListAsync(serviceName);

    List<KeyValuePair<string, int>> result = new List<KeyValuePair<string, int>>();

    foreach (Partition partition in partitions)
    {
        string proxyUrl =
            $"{proxyAddress}/api/VoteData?PartitionKey={{(Int64RangePartitionInformation)
partition.PartitionInformation).LowKey}}&PartitionKind=Int64Range";

        using (HttpResponseMessage response = await this.httpClient.GetAsync(proxyUrl))
        {
            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                continue;
            }

            result.AddRange(JsonConvert.DeserializeObject<List<KeyValuePair<string, int>>>(await
response.Content.ReadAsStringAsync()));
        }
    }

    return this.Json(result);
}

// PUT: api/Votes/name
[HttpPut("{name}")]
public async Task<IActionResult> Put(string name)
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);
    long partitionKey = this.GetPartitionKey(name);
    string proxyUrl = $"{proxyAddress}/api/VoteData/{name}?PartitionKey=
{partitionKey}&PartitionKind=Int64Range";

    StringContent putContent = new StringContent($"{{ 'name' : '{name}' }}", Encoding.UTF8,
"application/json");
    putContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

    using (HttpResponseMessage response = await this.httpClient.PutAsync(proxyUrl, putContent))
    {
        return new ContentResult()
        {
            StatusCode = (int) response.StatusCode,
            Content = await response.Content.ReadAsStringAsync()
        };
    }
}

// DELETE: api/Votes/name
[HttpDelete("{name}")]
public async Task<IActionResult> Delete(string name)
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);
    long partitionKey = this.GetPartitionKey(name);
    string proxyUrl = $"{proxyAddress}/api/VoteData/{name}?PartitionKey=
{partitionKey}&PartitionKind=Int64Range";

    using (HttpResponseMessage response = await this.httpClient.DeleteAsync(proxyUrl))
    {
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {

```

```

        return this.StatusCode((int) response.StatusCode);
    }
}

return new OkResult();
}

/// <summary>
/// Constructs a reverse proxy URL for a given service.
/// Example: http://localhost:19081/VotingApplication/VotingData/
/// </summary>
/// <param name="serviceName"></param>
/// <returns></returns>
private Uri GetProxyAddress(Uri serviceName)
{
    return new Uri($"http://localhost:19081{serviceName.AbsolutePath}");
}

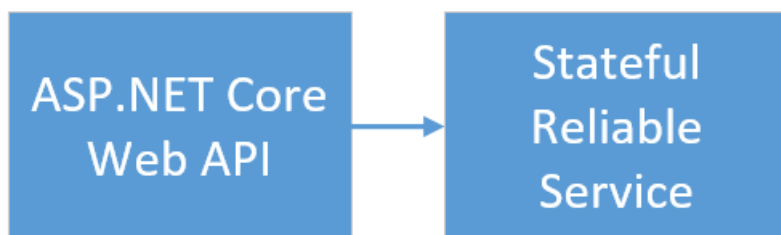
/// <summary>
/// Creates a partition key from the given name.
/// Uses the zero-based numeric position in the alphabet of the first letter of the name (0-25).
/// </summary>
/// <param name="name"></param>
/// <returns></returns>
private long GetPartitionKey(string name)
{
    return Char.ToUpper(name.First()) - 'A';
}
}

```

Walk through the voting sample application

The voting application consists of two services:

- Web front-end service (VotingWeb)- An ASP.NET Core web front-end service, which serves the web page and exposes web APIs to communicate with the backend service.
- Back-end service (VotingData)- An ASP.NET Core web service, which exposes an API to store the vote results in a reliable dictionary persisted on disk.



When you vote in the application the following events occur:

1. A JavaScript sends the vote request to the web API in the web front-end service as an HTTP PUT request.
2. The web front-end service uses a proxy to locate and forward an HTTP PUT request to the back-end service.
3. The back-end service takes the incoming request, and stores the updated result in a reliable dictionary, which gets replicated to multiple nodes within the cluster and persisted on disk. All the application's data is stored in the cluster, so no database is needed.

Debug in Visual Studio

When debugging application in Visual Studio, you are using a local Service Fabric development cluster. You have the option to adjust your debugging experience to your scenario. In this application, store data in the back-end service using a reliable dictionary. Visual Studio removes the application per default when you stop the debugger. Removing the application causes the data in the back-end service to also be removed. To persist the data between debugging sessions, you can change the **Application Debug Mode** as a property on the **Voting** project in Visual Studio.

To look at what happens in the code, complete the following steps:

1. Open the **VotingWeb\VotesController.cs** file and set a breakpoint in the web API's **Put** method (line 72).
2. Open the **VotingData\VoteDataController.cs** file and set a breakpoint in this web API's **Put** method (line 54).
3. Press **F5** to start the application in debug mode.
4. Go back to the browser and click a voting option or add a new voting option. You hit the first breakpoint in the web front-end's api controller.
 - a. This is where the JavaScript in the browser sends a request to the web API controller in the front-end service.

```
public async Task<IActionResult> Put(string name)
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);
    long partitionKey = this.GetPartitionKey(name);
    1 string proxyUrl = $"{proxyAddress}/api/VoteData/{name}?PartitionKey={partitionKey}&PartitionKind=Int64Range";

    StringContent putContent = new StringContent($"{{ 'name': '{name}' }}", Encoding.UTF8, "application/json");
    putContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

    2 using (HttpResponseMessage response = await this.httpClient.PutAsync(proxyUrl, putContent))
    {
        3 return new ContentResult()
        {
            StatusCode = (int)response.StatusCode,
            Content = await response.Content.ReadAsStringAsync()
        };
    }
}
```

- b. First construct the URL to the ReverseProxy for the back-end service (1).
 - c. Then send the HTTP PUT Request to the ReverseProxy (2).
 - d. Finally the return the response from the back-end service to the client (3).
5. Press **F5** to continue.
 - a. You are now at the break point in the back-end service.

```
public async Task<IActionResult> Put(string name)
{
    1 IReliableDictionary<string, int> votesDictionary = await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");
    2 using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        3 await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
        await tx.CommitAsync();
    }

    return new OkResult();
}
```

- b. In the first line in the method (1) use the `StateManager` to get or add a reliable dictionary called `counts`.
- c. All interactions with values in a reliable dictionary require a transaction, this using statement (2) creates that transaction.
- d. In the transaction, update the value of the relevant key for the voting option and commits the

operation (3). Once the commit method returns, the data is updated in the dictionary and replicated to other nodes in the cluster. The data is now safely stored in the cluster, and the back-end service can fail over to other nodes, still having the data available.

6. Press **F5** to continue.

To stop the debugging session, press **Shift+F5**.

Next steps

In this part of the tutorial, you learned how to:

- Create an ASP.NET Core Web API service as a stateful reliable service
- Create an ASP.NET Core Web Application service as a stateless web service
- Use the reverse proxy to communicate with the stateful service

Advance to the next tutorial:

[Deploy the application to Azure](#)

Quickstart: Deploy your first Azure Spring Cloud application

6/28/2021 • 11 minutes to read • [Edit Online](#)

This quickstart explains how to deploy a simple Azure Spring Cloud microservice application to run on Azure.

NOTE

Steeltoe support for Azure Spring Cloud is currently offered as a public preview. Public preview offerings allow customers to experiment with new features prior to their official release. Public preview features and services are not meant for production use. For more information about support during previews, see the [FAQ](#) or file a [Support request](#).

By following this quickstart, you'll learn how to:

- Generate a basic Steeltoe .NET Core project
- Provision an Azure Spring Cloud service instance
- Build and deploy the app with a public endpoint
- Stream logs in real time

The application code used in this quickstart is a simple app built with a .NET Core Web API project template. When you've completed this example, the application will be accessible online and can be managed via the Azure portal and the Azure CLI.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- [.NET Core 3.1 SDK](#). The Azure Spring Cloud service supports .NET Core 3.1 and later versions.
- [Azure CLI version 2.0.67 or later](#).
- [Git](#).

Install Azure CLI extension

Verify that your Azure CLI version is 2.0.67 or later:

```
az --version
```

Install the Azure Spring Cloud extension for the Azure CLI using the following command:

```
az extension add --name spring-cloud
```

Log in to Azure

1. Log in to the Azure CLI

```
az login
```

2. If you have more than one subscription, choose the one you want to use for this quickstart.


```
az account list -o table
```

```
az account set --subscription <Name or ID of a subscription from the last step>
```

Generate a Steeltoe .NET Core project

In Visual Studio, create an ASP.NET Core Web application named as "hello-world" with API project template. Please notice there will be a auto generated WeatherForecastController which will be our test endpoint later on.

1. Create a folder for the project source code and generate the project.

```
mkdir source-code
```

```
cd source-code
```

```
dotnet new webapi -n hello-world --framework netcoreapp3.1
```

2. Navigate into the project directory.

```
cd hello-world
```

3. Edit the *appSettings.json* file to add the following settings:

```
"spring": {  
  "application": {  
    "name": "hello-world"  
  }  
},  
"eureka": {  
  "client": {  
    "shouldFetchRegistry": true,  
    "shouldRegisterWithEureka": true  
  }  
}
```

4. Also in *appsettings.json*, change the log level for the `Microsoft` category from `Warning` to `Information`. This change ensures that logs will be produced when you view streaming logs in a later step.

The *appsettings.json* file now looks similar to the following example:

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Information",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "spring": {
    "application": {
      "name": "hello-world"
    }
  },
  "eureka": {
    "client": {
      "shouldFetchRegistry": true,
      "shouldRegisterWithEureka": true
    }
  }
}

```

5. Add dependencies and a `Zip` task to the `.csproj` file:

```

<ItemGroup>
  <PackageReference Include="Steeltoe.Discovery.ClientCore" Version="3.0.0" />
  <PackageReference Include="Microsoft.Azure.SpringCloud.Client" Version="2.0.0-preview.1" />
</ItemGroup>
<Target Name="Publish-Zip" AfterTargets="Publish">
  <ZipDirectory SourceDirectory="$(PublishDir)"
  DestinationFile="$(MSBuildProjectDirectory)/deploy.zip" Overwrite="true" />
</Target>

```

The packages are for Steeltoe Service Discovery and the Azure Spring Cloud client library. The `zip` task is for deployment to Azure. When you run the `dotnet publish` command, it generates the binaries in the `publish` folder, and this task zips the `publish` folder into a `.zip` file that you upload to Azure.

6. In the `Program.cs` file, add a `using` directive and code that uses the Azure Spring Cloud client library:

```
using Microsoft.Azure.SpringCloud.Client;
```

```

public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .UseAzureSpringCloudService()
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });

```

7. In the `Startup.cs` file, add a `using` directive and code that uses the Steeltoe Service Discovery at the end of the `ConfigureServices` and `Configure` methods:

```
using Steeltoe.Discovery.Client;
```

```
public void ConfigureServices(IServiceCollection services)
{
    // Template code not shown.

    services.AddDiscoveryClient(Configuration);
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // Template code not shown.

    app.UseDiscoveryClient();
}
```

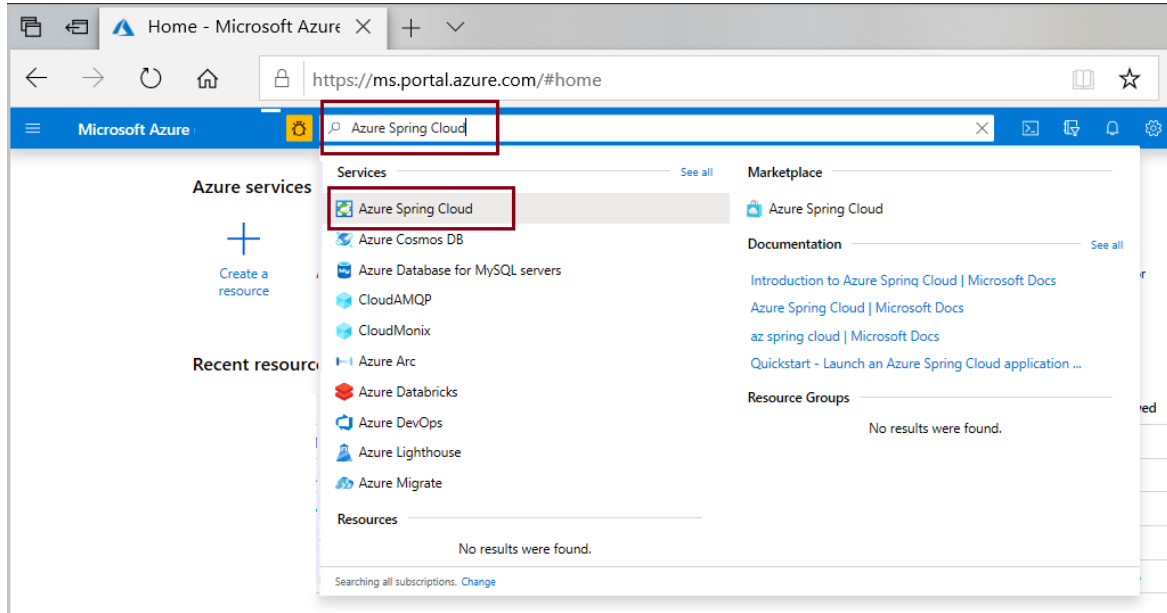
8. Build the project to make sure there are no compile errors.

```
dotnet build
```

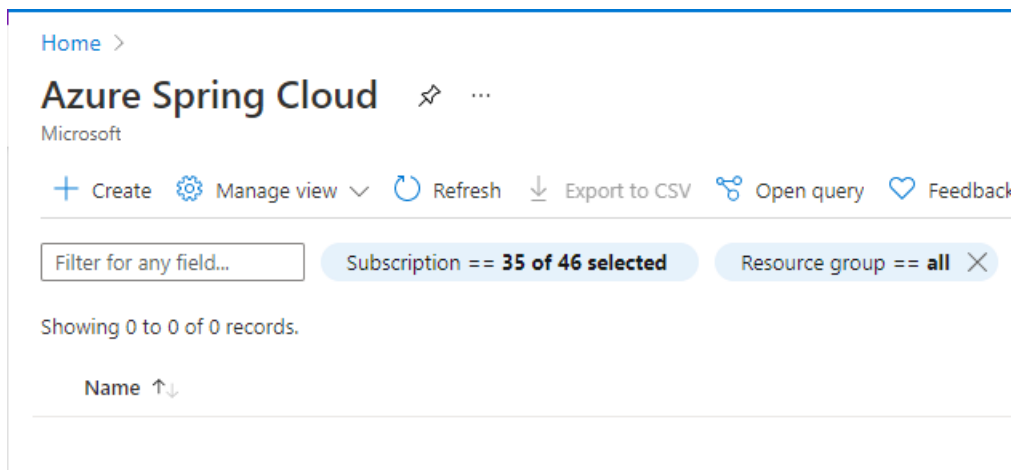
Provision a service instance

The following procedure creates an instance of Azure Spring Cloud using the Azure portal.

1. Open the [Azure portal](#).
2. From the top search box, search for *Azure Spring Cloud*.
3. Select *Azure Spring Cloud* from the results.

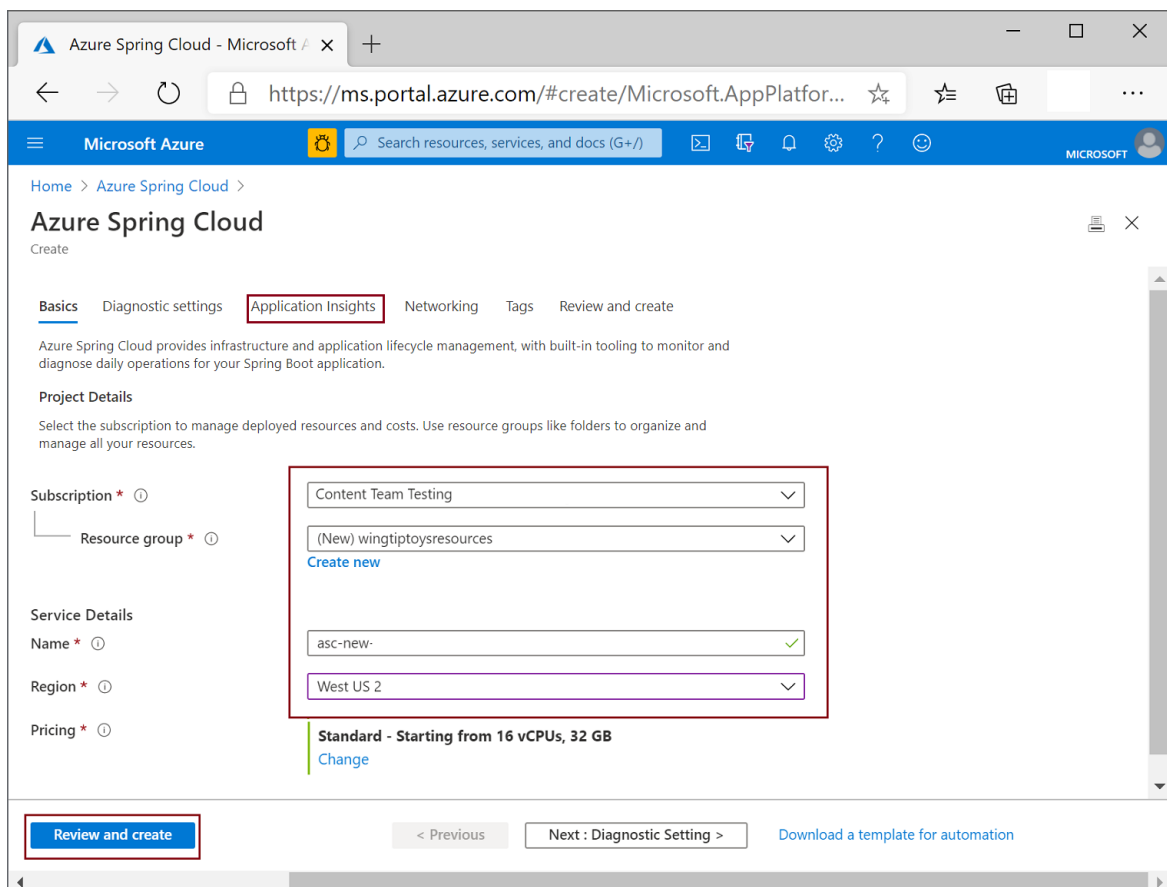


4. On the Azure Spring Cloud page, select **+ Create**.



5. Fill out the form on the Azure Spring Cloud **Create** page. Consider the following guidelines:

- **Subscription:** Select the subscription you want to be billed for this resource.
- **Resource group:** Create a new resource group. The name you enter here will be used in later steps as `<resource group name>`.
- **Service Details/Name:** Specify the `<service instance name>`. The name must be between 4 and 32 characters long and can contain only lowercase letters, numbers, and hyphens. The first character of the service name must be a letter and the last character must be either a letter or a number.
- **Region:** Select the region for your service instance.



6. Select **Review and create**.

7. Select **Create**.

Build and deploy the app

The following procedure builds and deploys the project that you created earlier.

1. Make sure the command prompt is still in the project folder.

2. Run the following command to build the project, publish the binaries, and store the binaries in a `.zip` file in the project folder.

```
dotnet publish -c release -o ./publish
```

3. Create an app in your Azure Spring Cloud instance with a public endpoint assigned. Use the same application name "hello-world" that you specified in `appsettings.json`.

```
az spring-cloud app create -n hello-world -s <service instance name> -g <resource group name> --assign-endpoint --runtime-version NetCore_31
```

4. Deploy the `.zip` file to the app.

```
az spring-cloud app deploy -n hello-world -s <service instance name> -g <resource group name> --runtime-version NetCore_31 --main-entry hello-world.dll --artifact-path ./deploy.zip
```

The `--main-entry` option identifies the `.dll` file that contains the application's entry point. After the service uploads the `.zip` file, it extracts all the files and folders and tries to execute the entry point in the `.dll` file specified by `--main-entry`.

It takes a few minutes to finish deploying the application. To confirm that it has deployed, go to the **Apps** blade in the Azure portal.

Test the app

Once deployment has completed, access the app at the following URL:

```
https://<service instance name>-hello-world.azuremicroservices.io/weatherforecast
```

The app returns JSON data similar to the following example:

```
[{"date": "2020-09-08T21:01:50.0198835+00:00", "temperatureC": 14, "temperatureF": 57, "summary": "Bracing"}, {"date": "2020-09-09T21:01:50.0200697+00:00", "temperatureC": -14, "temperatureF": 7, "summary": "Bracing"}, {"date": "2020-09-10T21:01:50.0200715+00:00", "temperatureC": 27, "temperatureF": 80, "summary": "Freezing"}, {"date": "2020-09-11T21:01:50.0200717+00:00", "temperatureC": 18, "temperatureF": 64, "summary": "Chilly"}, {"date": "2020-09-12T21:01:50.0200719+00:00", "temperatureC": 16, "temperatureF": 60, "summary": "Chilly"}]
```

Stream logs in real time

Use the following command to get real time logs from the App.

```
az spring-cloud app logs -n hello-world -s <service instance name> -g <resource group name> --lines 100 -f
```

Logs appear in the output:

```
[Azure Spring Cloud] The following environment variables are loaded:
2020-09-08 20:58:42,432 INFO supervisor started with pid 1
2020-09-08 20:58:43,435 INFO spawned: 'event-gather_00' with pid 9
2020-09-08 20:58:43,436 INFO spawned: 'dotnet-app_00' with pid 10
2020-09-08 20:58:43 [Warning] No managed processes are running. Wait for 30 seconds...
2020-09-08 20:58:44,843 INFO success: event-gather_00 entered RUNNING state, process has stayed up for >
than 1 seconds (startsecs)
2020-09-08 20:58:44,843 INFO success: dotnet-app_00 entered RUNNING state, process has stayed up for > than
1 seconds (startsecs)
←[40m←[32minfo←[39m←[22m←[49m: Steeltoe.Discovery.Eureka.DiscoveryClient[0]
  Starting HeartBeat
info: Microsoft.Hosting.Lifetime[0]
  Now listening on: http://[::]:1025
info: Microsoft.Hosting.Lifetime[0]
  Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
  Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
  Content root path: /netcorepublish/6e4db42a-b160-4b83-a771-c91adec18c60
2020-09-08 21:00:13 [Information] [10] Start listening...
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
  Request starting HTTP/1.1 GET http://asc-svc-hello-world.azuremicroservices.io/weatherforecast
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[0]
  Executing endpoint 'hello_world.Controllers.WeatherForecastController.Get (hello-world)'
info: Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker[3]
  Route matched with {action = "Get", controller = "WeatherForecast"}. Executing controller action with
signature System.Collections.Generic.IEnumerable`1[hello_world.WeatherForecast] Get() on controller
hello_world.Controllers.WeatherForecastController (hello-world).
info: Microsoft.AspNetCore.Mvc.Infrastructure.ObjectResultExecutor[1]
  Executing ObjectResult, writing value of type 'hello_world.WeatherForecast[]'.
info: Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker[2]
  Executed action hello_world.Controllers.WeatherForecastController.Get (hello-world) in 1.8902ms
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[1]
  Executed endpoint 'hello_world.Controllers.WeatherForecastController.Get (hello-world)'
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
  Request finished in 4.2591ms 200 application/json; charset=utf-8
```

TIP

Use `az spring-cloud app logs -h` to explore more parameters and log stream functionalities.

For advanced log analytics features, visit **Logs** tab in the menu on [Azure portal](#). Logs here have a latency of a few minutes.

The screenshot shows the Azure portal interface for viewing logs. The top navigation bar includes 'Home > Microsoft.AppPlatform | Overview >'. The main header is '<Service Instance Name> | Logs'. Below the header, there's a search bar and a 'New Query' button. The query editor contains the following Kusto query:

```
// Show the application logs which contain the "error" or "exception" terms
// Show the application logs which contain the "error" or "exception" terms in the last hour.
AppPlatformLogsForSpring
| where TimeGenerated > ago(1h)
| project TimeGenerated, ServiceName, AppName, InstanceName, Log
```

The results table shows the following data:

TimeGenerated [UTC]	ServiceName	AppName	InstanceName	Log
8/5/2020, 7:30:17.474 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:30:17.474 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:45:17.476 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:45:17.476 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:50:17.476 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:50:17.476 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:55:17.477 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:55:17.477 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:05:17.471 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:05:17.471 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:10:17.472 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:10:17.471 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:15:17.479 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:15:17.472 INFO [demo...] 1 --- [trap-executor-0] c.n.d
8/5/2020, 7:20:17.473 AM	serviceName	demo	demo-default-4-59cc9b54d-vf856	2020-08-05 07:20:17.472 INFO [demo...] 1 --- [trap-executor-0] c.n.d

The bottom of the screen shows pagination: 'Page 1 of 1', '50 Items per page', and '1 - 11 of 11 items'.

This quickstart explains how to deploy a simple Azure Spring Cloud microservice application to run on Azure.

The application code used in this tutorial is a simple app built with Spring Initializr. When you've completed this example, the application will be accessible online and can be managed via the Azure portal.

This quickstart explains how to:

- Generate a basic Spring Cloud project
- Provision a service instance
- Build and deploy the app with a public endpoint
- Stream logs in real time

Prerequisites

To complete this quickstart:

- [Install JDK 8](#)
- [Sign up for an Azure subscription](#)
- (Optional) [Install the Azure CLI version 2.0.67 or higher](#) and the Azure Spring Cloud extension with command: `az extension add --name spring-cloud`
- (Optional) [Install the Azure Toolkit for IntelliJ](#) and [sign-in](#)

Generate a Spring Cloud project

Start with [Spring Initializr](#) to generate a sample project with recommended dependencies for Azure Spring Cloud. The following image shows the Initializr set up for this sample project.

```
https://start.spring.io#!type=maven-project&language=java&platformVersion=2.3.12.RELEASE&packaging=jar&jvmVersion=1.8&groupId=com.example&artifactId=hellospring&name=hellospring&description=Demo%20project%20for%20Spring%20Boot&packageName=com.example.hellospring&dependencies=web,cloud-eureka,actuator,cloud-starter-sleuth,cloud-starter-zipkin,cloud-config-client
```

Note that this example uses Java version 8. If you want to use Java version 11, change the option under **Project Metadata**.

Project

Maven Project
 Gradle Project

Language

Java Kotlin
 Groovy

Spring Boot

2.4.0 (SNAPSHOT) 2.4.0 (M4)
 2.3.5 (SNAPSHOT) 2.3.4 2.2.11 (SNAPSHOT)
 2.2.10 2.1.18 (SNAPSHOT) 2.1.17

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging Jar War

Java 15 11 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Eureka Discovery Client SPRING CLOUD DISCOVERY

a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

Spring Boot Actuator OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Sleuth SPRING CLOUD TRACING

Distributed tracing via logs with Spring Cloud Sleuth.

Zipkin Client SPRING CLOUD TRACING

Distributed tracing with an existing Zipkin installation and Spring Cloud Sleuth Zipkin.

Config Client SPRING CLOUD CONFIG

Client that connects to a Spring Cloud Config Server to fetch the application's configuration.

GENERATE CTRL + G
EXPLORE CTRL + SPACE
SHARE...

1. Click **Generate** when all the dependencies are set. Download and unpack the package, then create a web controller for a simple web application by adding `src/main/java/com/example/hellospring/HelloController.java` as follows:

```

package com.example.hellospring;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Azure Spring Cloud!";
    }

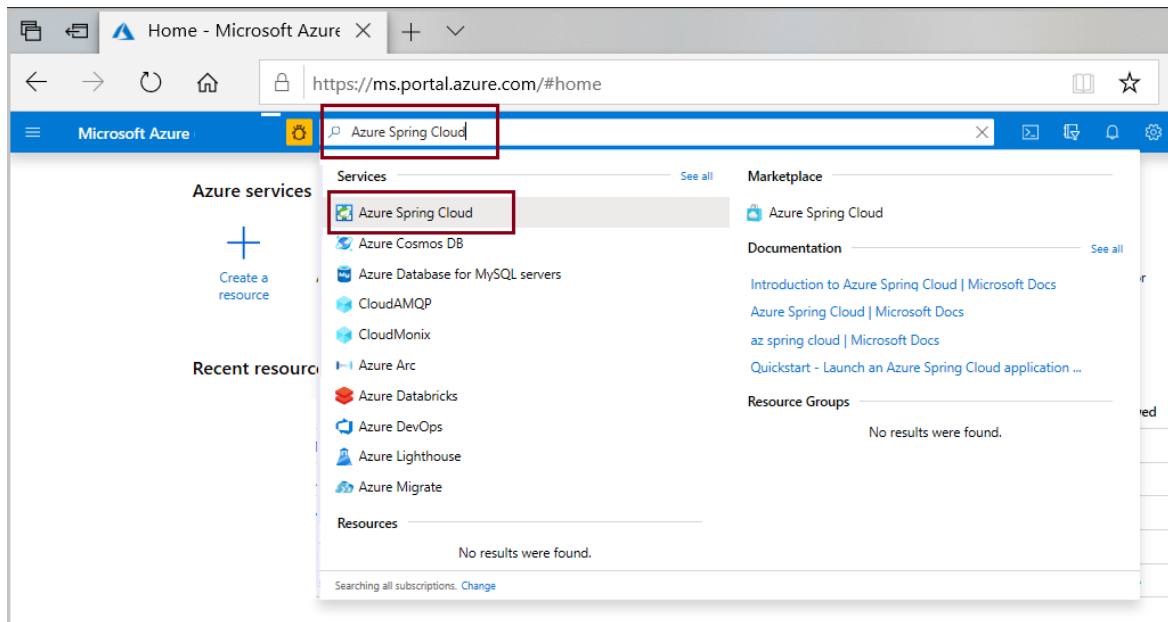
}

```

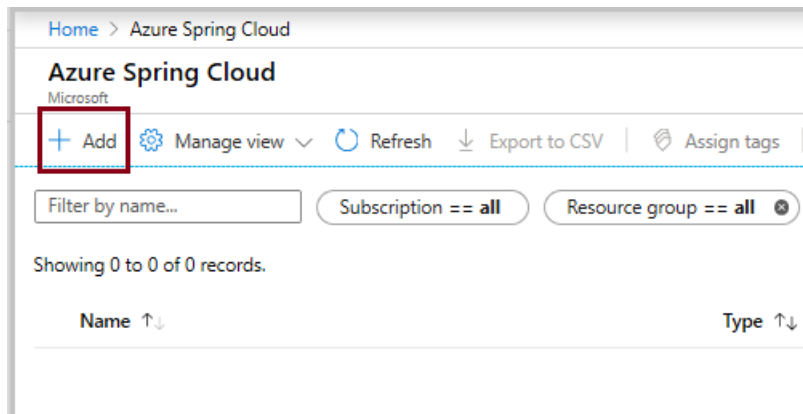
Provision an instance of Azure Spring Cloud

The following procedure creates an instance of Azure Spring Cloud using the Azure portal.

1. In a new tab, open the [Azure portal](#).
2. From the top search box, search for *Azure Spring Cloud*.
3. Select *Azure Spring Cloud* from the results.

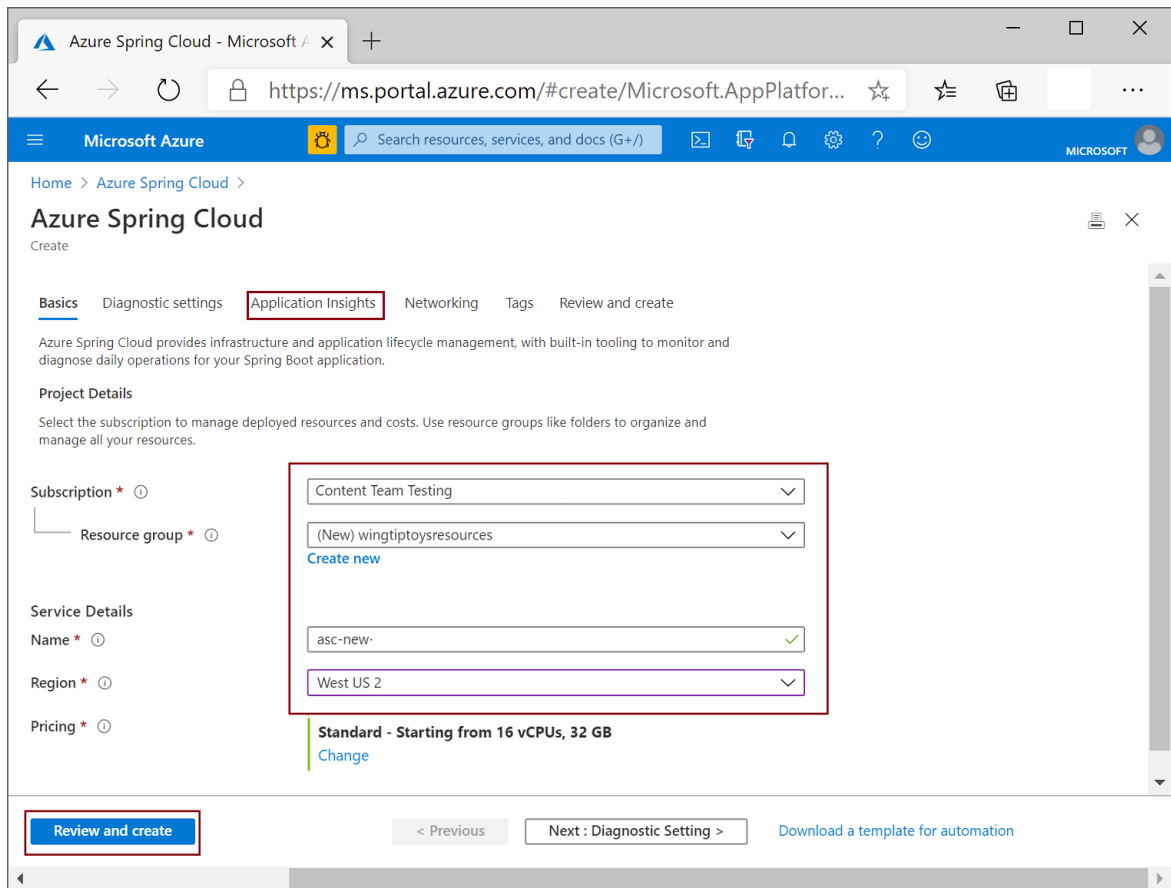


4. On the Azure Spring Cloud page, click + Create.



5. Fill out the form on the Azure Spring Cloud Create page. Consider the following guidelines:

- **Subscription:** Select the subscription you want to be billed for this resource.
- **Resource group:** Creating new resource groups for new resources is a best practice. This will be used in later steps as `<resource group name>`.
- **Service Details/Name:** Specify the `<service instance name>`. The name must be between 4 and 32 characters long and can contain only lowercase letters, numbers, and hyphens. The first character of the service name must be a letter and the last character must be either a letter or a number.
- **Location:** Select the region for your service instance.



6. Click **Review and create**.

Build and deploy the app

- [CLI](#)
- [IntelliJ](#)

The following procedure builds and deploys the application using the Azure CLI. Execute the following command at the root of the project.

1. Build the project using Maven:

```
mvn clean package -DskipTests
```

2. (If you haven't already installed it) Install the Azure Spring Cloud extension for the Azure CLI:

```
az extension add --name spring-cloud
```

3. Create the app with public endpoint assigned. If you selected Java version 11 when generating the Spring Cloud project, include the `--runtime-version=Java_11` switch.

```
az spring-cloud app create -n helloworld -s <service instance name> -g <resource group name> --assign-endpoint true
```

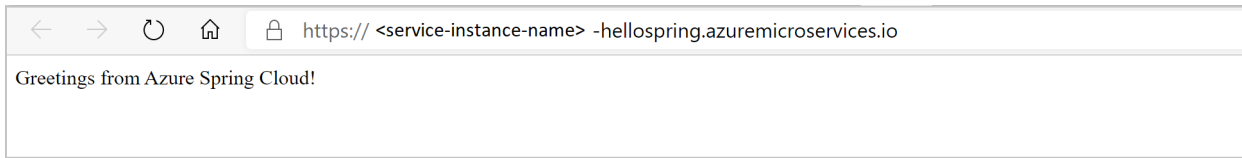
4. Deploy the Jar file for the app (`target\helloworld-0.0.1-SNAPSHOT.jar` on Windows):

```
az spring-cloud app deploy -n helloworld -s <service instance name> -g <resource group name> --jar-path <jar file path>
```

- It takes a few minutes to finish deploying the application. To confirm that it has deployed, go to the **Apps** blade in the Azure portal. You should see the status of the application.

Once deployment has completed, you can access the app at

```
https://<service instance name>-hellospring.azuremicroservices.io/.
```



Streaming logs in real time

- [CLI](#)
- [IntelliJ](#)

Use the following command to get real time logs from the App.

```
az spring-cloud app logs -n hellospring -s <service instance name> -g <resource group name> --lines 100 -f
```

Logs appear in the results:

```
PS C:\Users\user\Documents\vsworks\hellospring> az spring-cloud app logs -n hellospring -s user qs2 -g user qs2
Command group 'spring-cloud' is in preview. It may be changed/removed in a future release.
=====|_|=====|_|_/_/_/
:: Spring Boot :: (v2.3.3.RELEASE)

2020-08-20 03:32:58.155 INFO [hellospring,,,] 1 --- [          main] c.e.hellospring.HellospringApplication : No active profile set, falling bac
k to default profiles: default
2020-08-20 03:33:04.030 WARN [hellospring,,,] 1 --- [          main] o.s.boot.actuate.endpoint.EndpointId : Endpoint ID 'service-registry' con
tains invalid characters, please migrate to a valid format.
2020-08-20 03:33:05.144 INFO [hellospring,,,] 1 --- [          main] o.s.cloud.context.scope.GenericScope : BeanFactory id=767d548c-c478-3c99-
a04c-9e27571f6a0c
2020-08-20 03:33:08.492 INFO [hellospring,,,] 1 --- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 1
025 (http)
2020-08-20 03:33:08.512 INFO [hellospring,,,] 1 --- [          main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-08-20 03:33:08.513 INFO [hellospring,,,] 1 --- [          main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache T
omcat/9.0.37]
2020-08-20 03:33:08.653 INFO [hellospring,,,] 1 --- [          main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebAp
```

TIP

Use `az spring-cloud app logs -h` to explore more parameters and log stream functionalities.

For advanced logs analytics features, visit **Logs** tab in the menu on [Azure portal](#). Logs here have a latency of a few minutes.

Clean up resources

In the preceding steps, you created Azure resources that will continue to accrue charges while they remain in your subscription. If you don't expect to need these resources in the future, delete the resource group from the portal or by running the following command in the Azure CLI:

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

Next steps

In this quickstart, you learned how to:

- Generate a basic Azure Spring Cloud project
- Provision a service instance
- Build and deploy the app with a public endpoint
- Stream logs in real time

To learn how to use more Azure Spring capabilities, advance to the quickstart series that deploys a sample application to Azure Spring Cloud:

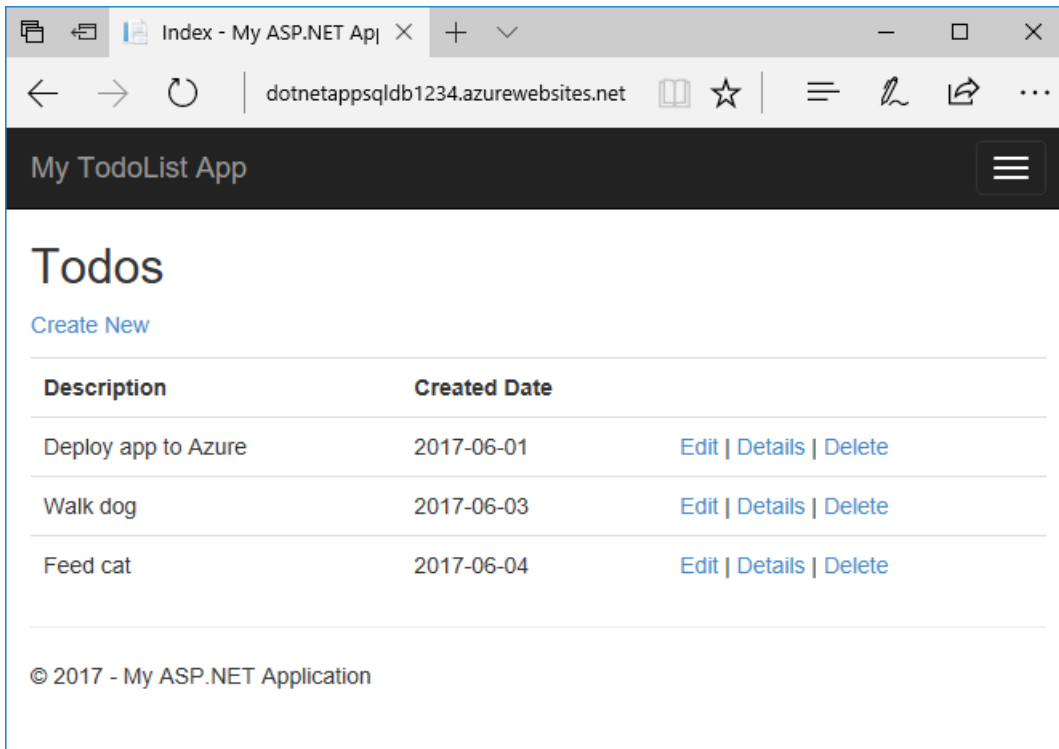
[Build and Run Microservices](#)

More samples are available on GitHub: [Azure Spring Cloud Samples](#).

Tutorial: Deploy an ASP.NET app to Azure with Azure SQL Database

3/19/2021 • 12 minutes to read • [Edit Online](#)

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



In this tutorial, you learn how to:

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

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

Prerequisites

To complete this tutorial:

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

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

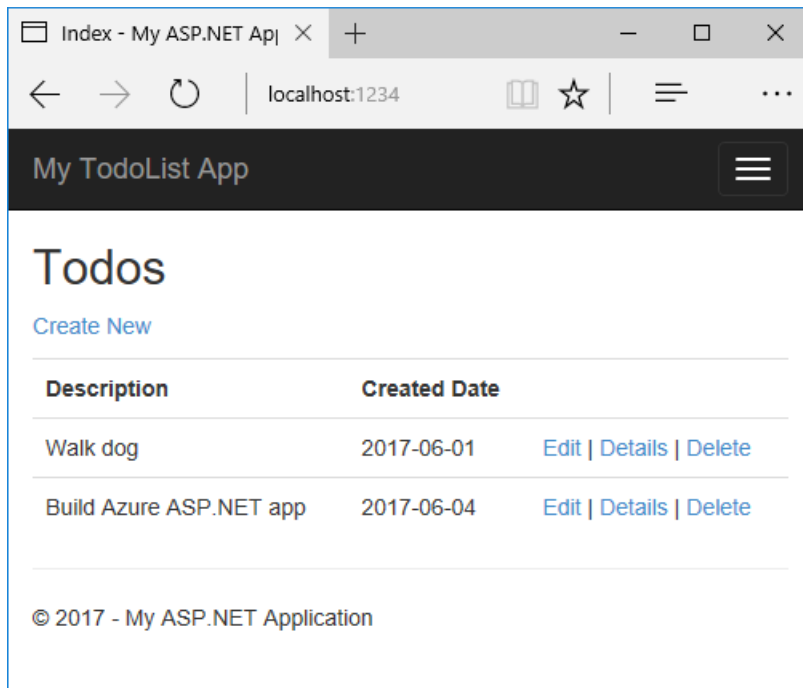
Download the sample

1. [Download the sample project](#).
2. Extract (unzip) the *dotnet-sqldb-tutorial-master.zip* file.

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

Run the app

1. Open the *dotnet-sqldb-tutorial-master/DotNetAppSqlDb.sln* file in Visual Studio.
2. Type `Ctrl+F5` to run the app without debugging. The app is displayed in your default browser.
3. Select the **Create New** link and create a couple *to-do* items.

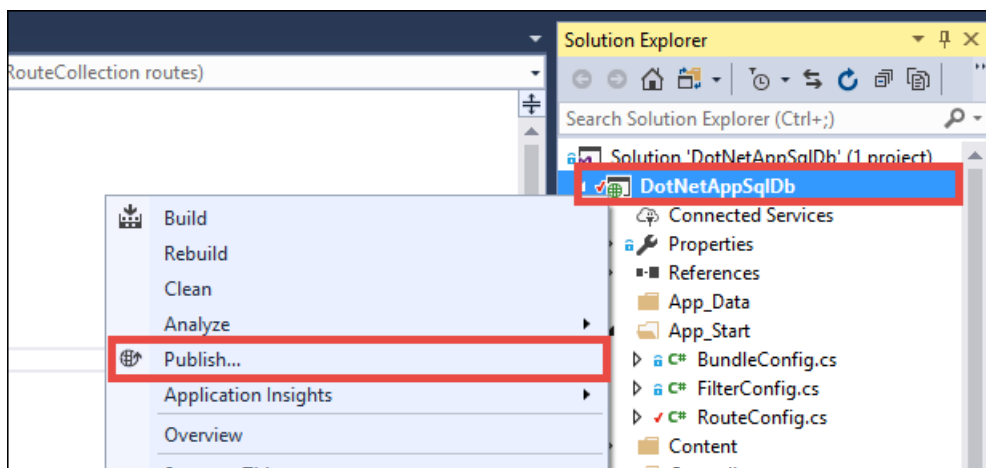


4. Test the **Edit**, **Details**, and **Delete** links.

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

Publish ASP.NET application to Azure

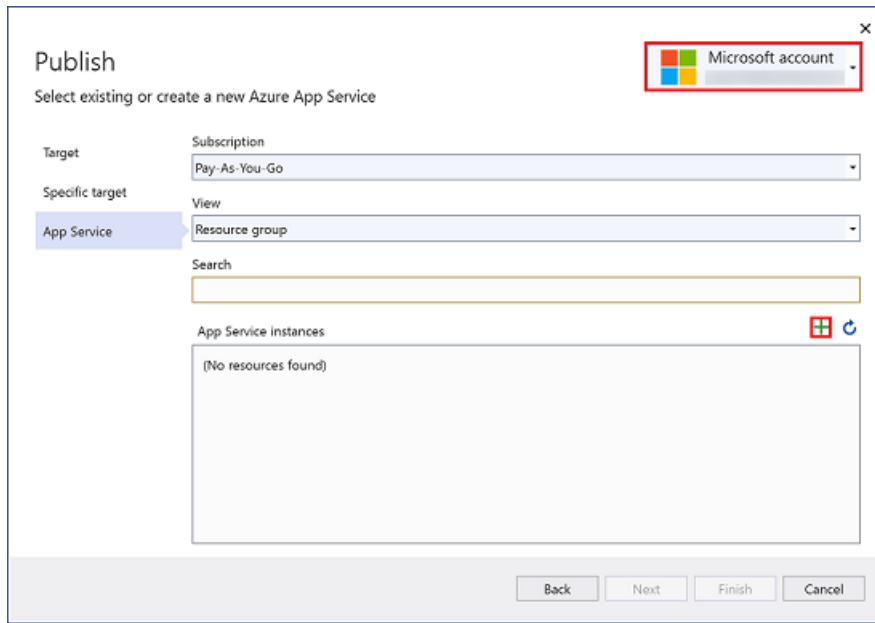
1. In the **Solution Explorer**, right-click your **DotNetAppSqlDb** project and select **Publish**.



2. Select **Azure** as your target and click **Next**.
3. Make sure that **Azure App Service (Windows)** is selected and click **Next**.

Sign in and add an app

1. In the **Publish** dialog, click **Add an account** from the account manager drop down.
2. Sign in to your Azure subscription. If you're already signed into a Microsoft account, make sure that account holds your Azure subscription. If the signed-in Microsoft account doesn't have your Azure subscription, click it to add the correct account.
3. In the **App Service instances** pane, click **+**.



Configure the web app name

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

NOTE

Don't select Create yet.

App Service (Windows) Create new

Microsoft account

Name: DotNetAppSqlDb20200923225505

Subscription: Pay-As-You-Go

Resource group: DotNetAppSqlDb-rg* [New...](#)

Hosting Plan: DotNetAppSqlDb20200923225505Plan* (South Central US, S1) [New...](#)

Export... ~~Create~~ Cancel

Create a resource group

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

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

App Service (Windows) Create new

Microsoft account

Name: DotNetAppSqlDb20200923225505

Subscription: Pay-As-You-Go

Resource group: DotNetAppSqlDb-rg* [New...](#)

Hosting Plan: DotNetAppSqlDb20200923225505Plan* (South Central US, S1) [New...](#)

Export... Create Cancel

2. Name the resource group **myResourceGroup**.

Create an App Service plan

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

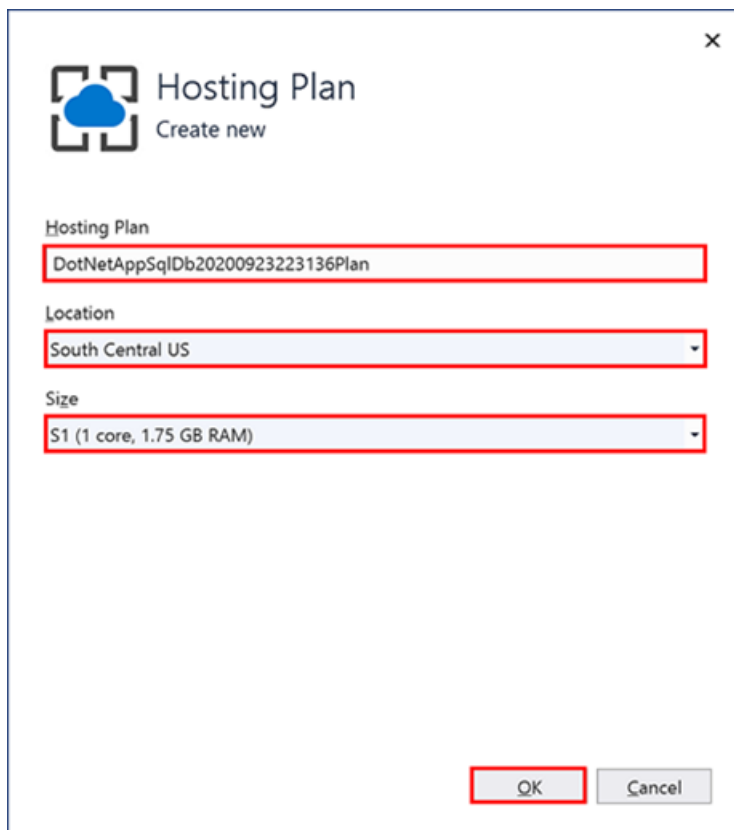
App Service plans define:

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

1. Next to **Hosting Plan**, click **New**.

2. In the **Configure App Service Plan** dialog, configure the new App Service plan with the following settings and click **OK**:

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



Hosting Plan
Create new

Hosting Plan
DotNetAppSqlDb20200923223136Plan

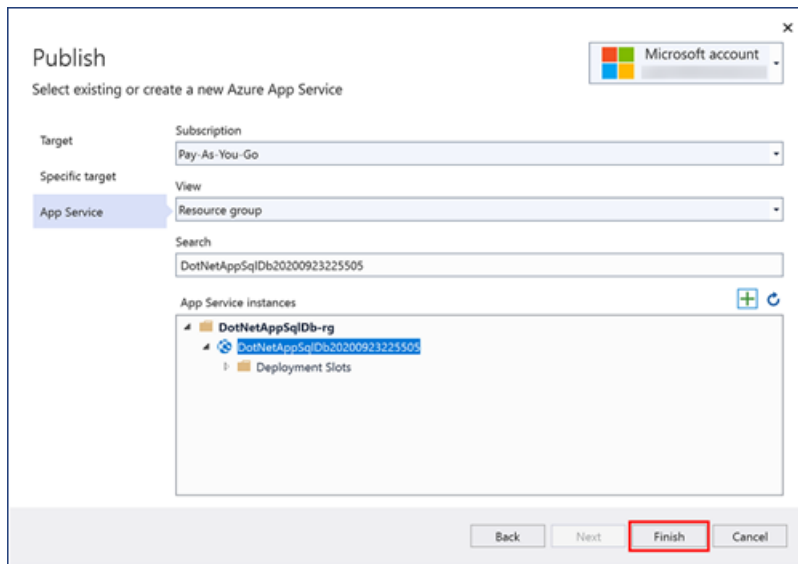
Location
South Central US

Size
S1 (1 core, 1.75 GB RAM)

OK Cancel

3. Click **Create** and wait for the Azure resources to be created.

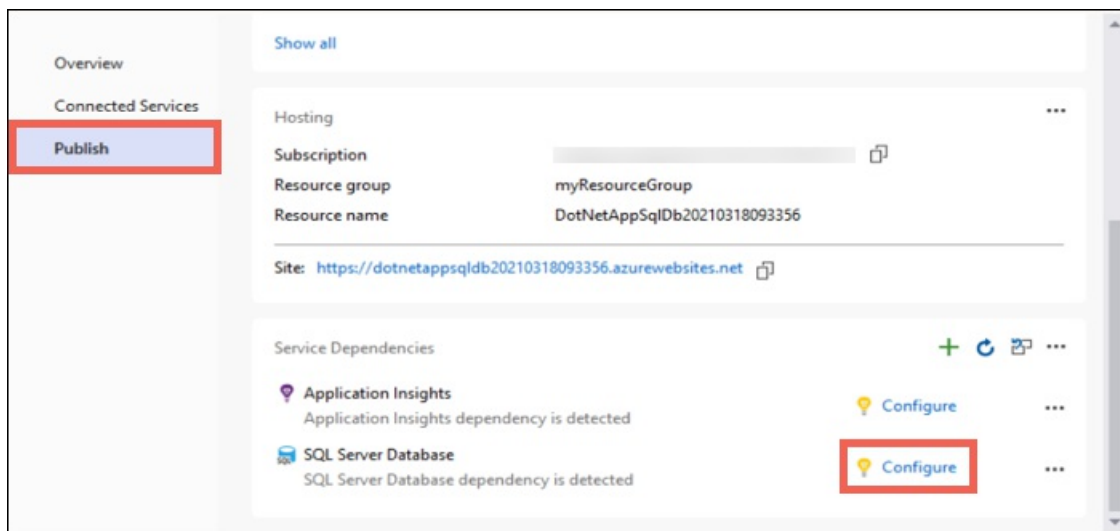
4. The **Publish** dialog shows the resources you've configured. Click **Finish**.



Create a server and database

Before creating a database, you need a [logical SQL server](#). A logical SQL server is a logical construct that contains a group of databases managed as a group.

1. In the **Publish** dialog, scroll down to the **Service Dependencies** section. Next to **SQL Server Database**, click **Configure**.



2. Select **Azure SQL Database** and click **Next**.
3. In the **Configure Azure SQL Database** dialog, click **+**.
4. Next to **Database server**, click **New**.

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

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

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

SQL Server
Create new

Database server name
dotnetappsql114dserver

Location
Central US

Administrator username
sqladmin

Administrator password
.....

Administrator password (confirm)
.....

OK Cancel

IMPORTANT

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

6. Click OK.
7. In the Azure SQL Database dialog, keep the default generated **Database Name**. Select **Create** and wait for the database resources to be created.

Azure SQL Database
Create new

Microsoft account

Database name
DotNetAppSqlDb_db

Subscription
Pay-As-You-Go

Resource group
DotNetAppSqlDb_rg New...

Database server
dotnetappsql114dserver* (Central US) New...

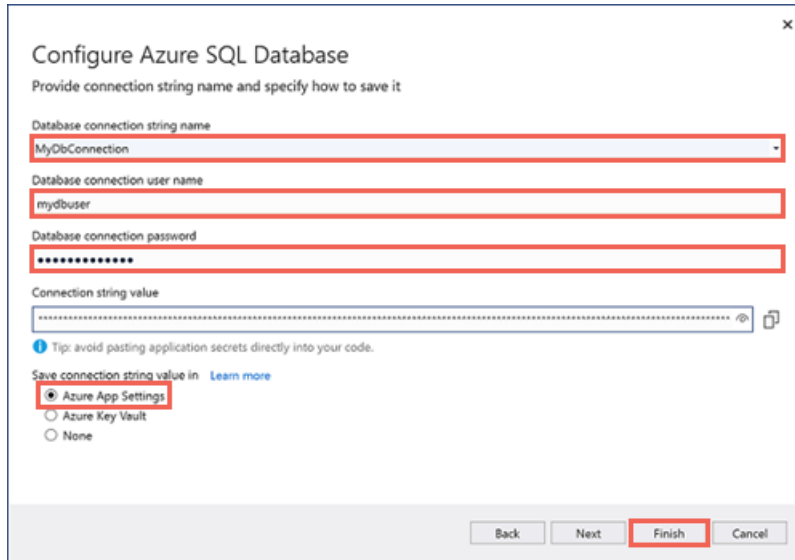
Database administrator username (must have permissions to create)
sqladmin

Database administrator password
.....

Export... Create Cancel

Configure database connection

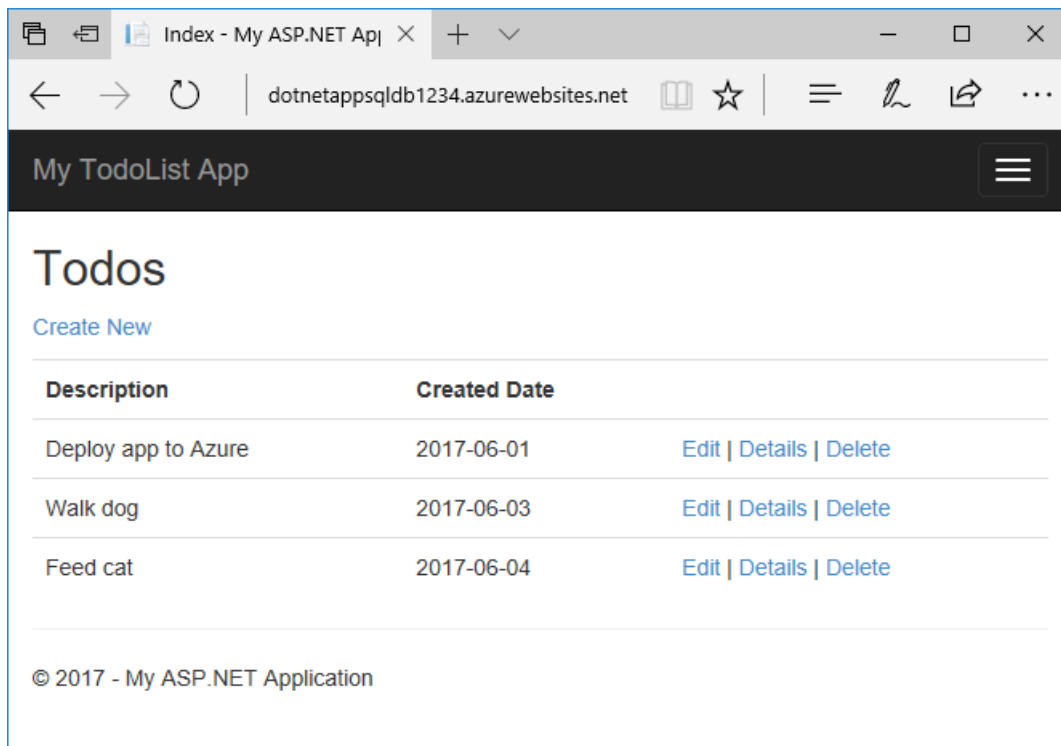
1. When the wizard finishes creating the database resources, click **Next**.
2. In the **Database connection string Name**, type *MyDbConnection*. This name must match the connection string that is referenced in *Models/MyDatabaseContext.cs*.
3. In **Database connection user name** and **Database connection password**, type the administrator username and password you used in [Create a server](#).
4. Make sure **Azure App Settings** is selected and click **Finish**.



5. Wait for configuration wizard to finish and click **Close**.

Deploy your ASP.NET app

1. In the **Publish** tab scroll back up to the top and click **Publish**. Once your ASP.NET app is deployed to Azure. Your default browser is launched with the URL to the deployed app.
2. Add a few to-do items.



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

Access the database locally

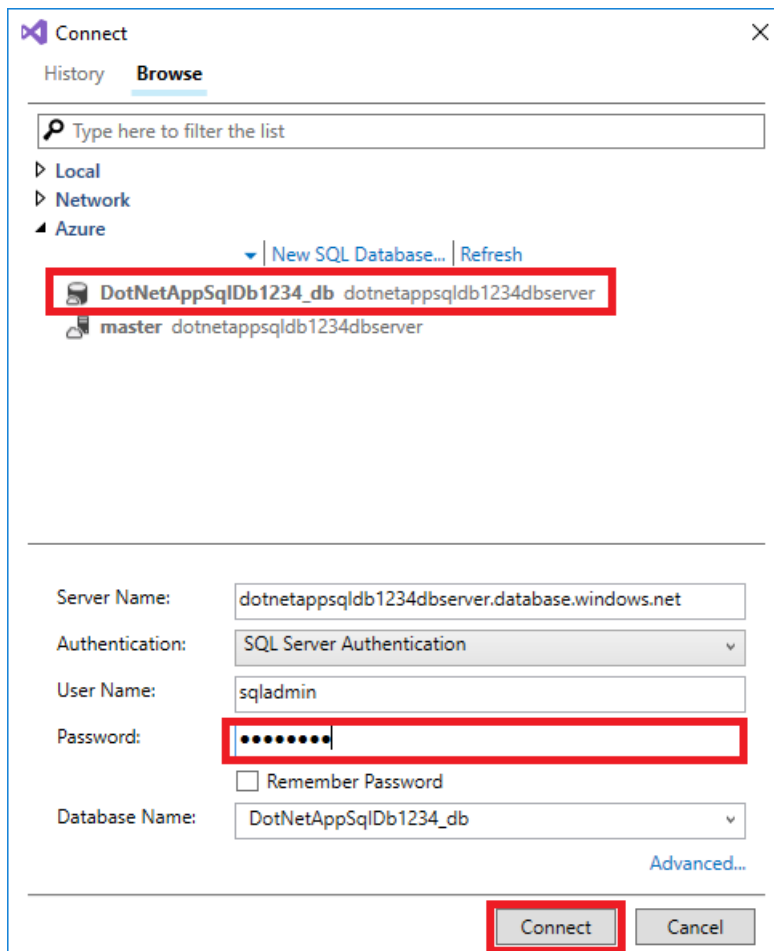
Visual Studio lets you explore and manage your new database in Azure easily in the **SQL Server Object Explorer**. The new database already opened its firewall to the App Service app that you created, but to access it from your local computer (such as from Visual Studio), you must open a firewall for your local machine's public IP address. If your internet service provider changes your public IP address, you need to reconfigure the firewall to access the Azure database again.

Create a database connection

1. From the **View** menu, select **SQL Server Object Explorer**.
2. At the top of **SQL Server Object Explorer**, click the **Add SQL Server** button.

Configure the database connection

1. In the **Connect** dialog, expand the **Azure** node. All your SQL Database instances in Azure are listed here.
2. Select the database that you created earlier. The connection you created earlier is automatically filled at the bottom.
3. Type the database administrator password you created earlier and click **Connect**.

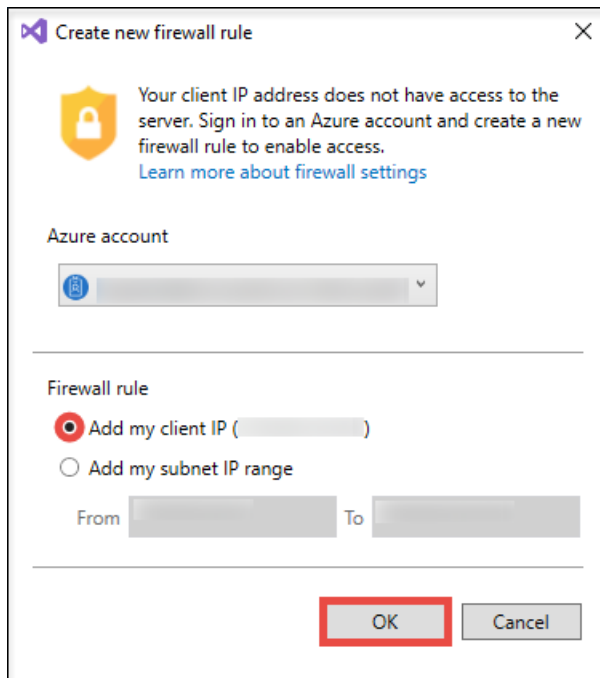


Allow client connection from your computer

The **Create a new firewall rule** dialog is opened. By default, a server only allows connections to its databases from Azure services, such as your Azure app. To connect to your database from outside of Azure, create a firewall rule at the server level. The firewall rule allows the public IP address of your local computer.

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

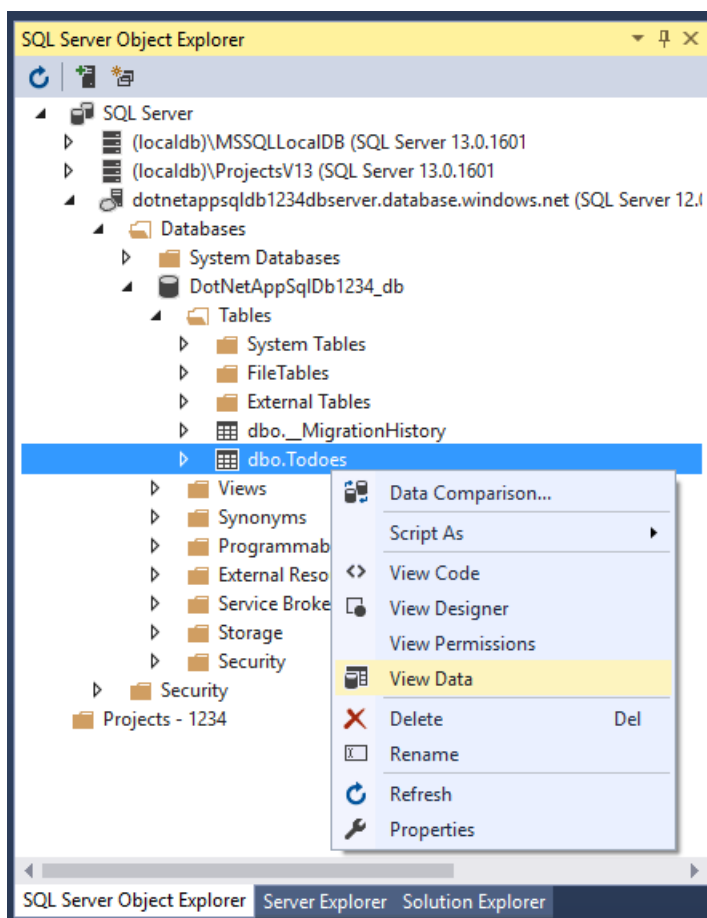
1. Make sure that **Add my client IP** is selected and click **OK**.



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

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

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



Update app with Code First Migrations

You can use the familiar tools in Visual Studio to update your database and app in Azure. In this step, you use

Code First Migrations in Entity Framework to make a change to your database schema and publish it to Azure.

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

Update your data model

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

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

Run Code First Migrations locally

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

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

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

```
Enable-Migrations
```

3. Add a migration:

```
Add-Migration AddProperty
```

4. Update the local database:

```
Update-Database
```

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

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

Use the new property

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

1. Open *Controllers\TodosController.cs*.

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

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

3. Open *Views\Todos\Create.cshtml*.

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

```

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

```

5. Open `Views\Todos\Index.cshtml`.

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

```

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

```

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

```

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

```

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

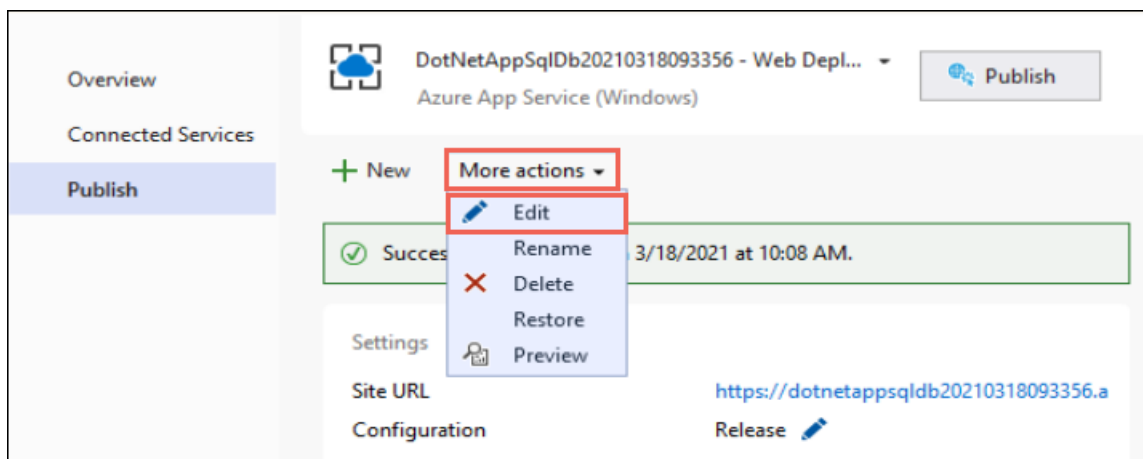
8. Type `ctr+F5` to run the app.

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

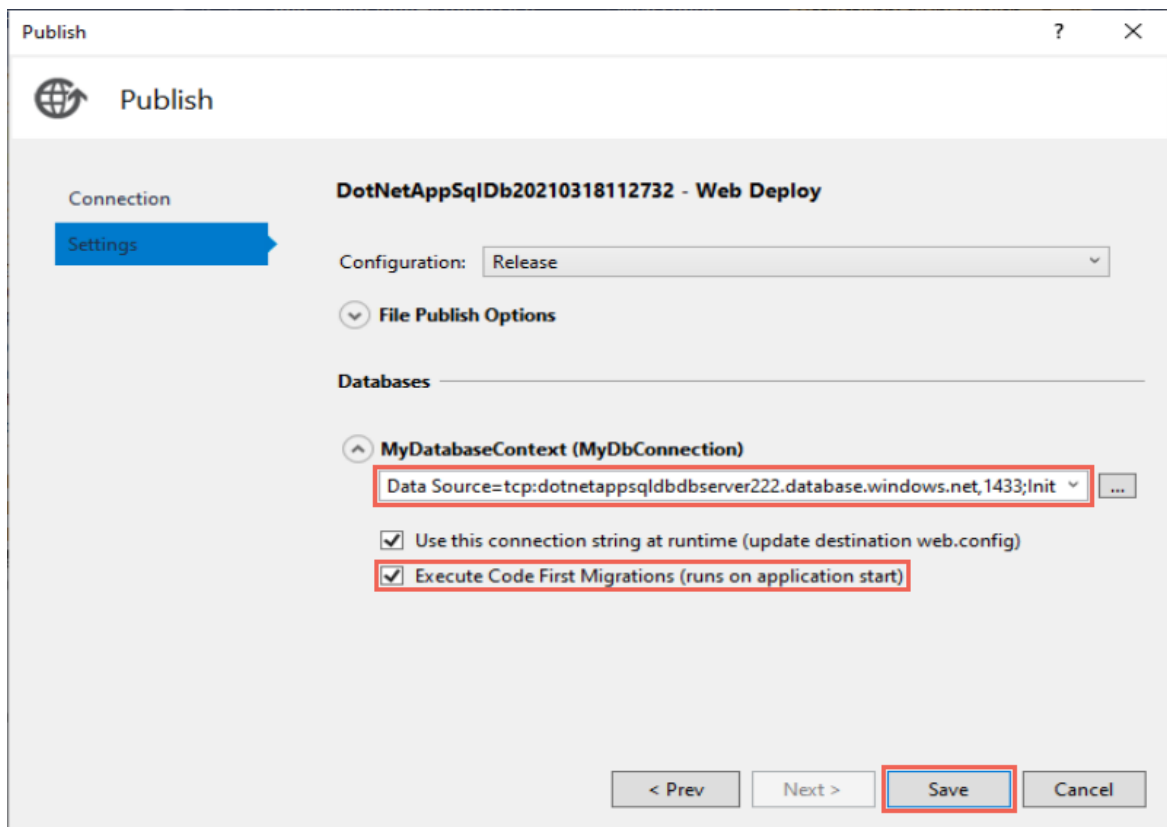
Enable Code First Migrations in Azure

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

1. Just like before, right-click your project and select **Publish**.
2. Click **More actions** > **Edit** to open the publish settings.



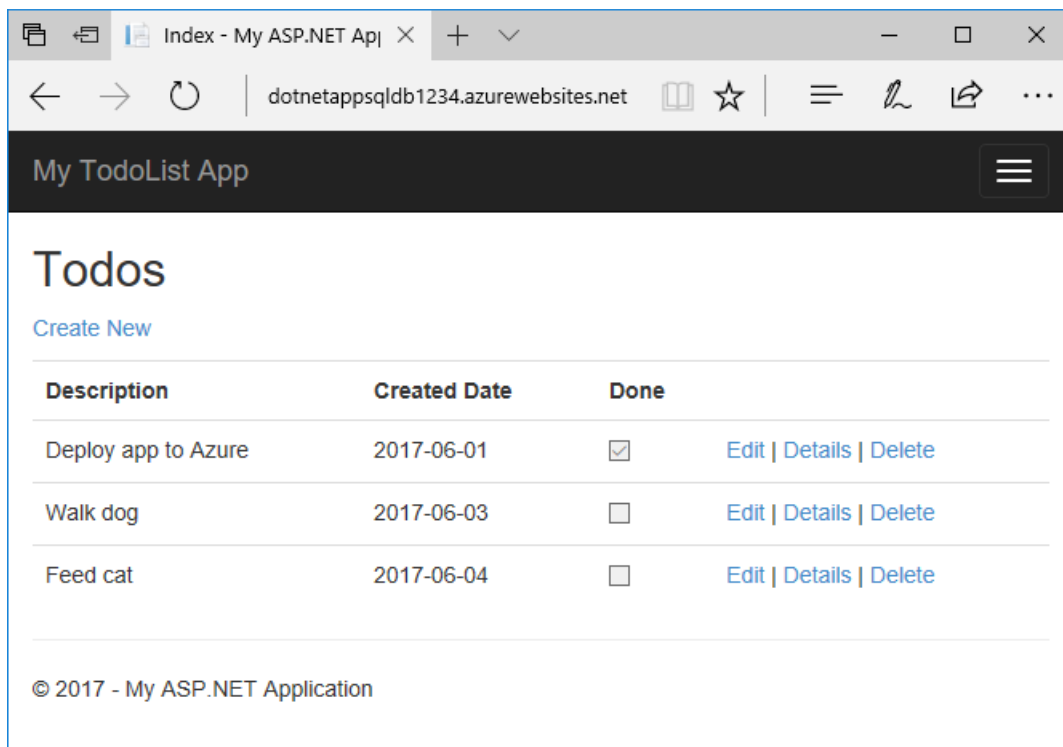
3. In the `MyDatabaseContext` dropdown, select the database connection for your Azure SQL Database.
4. Select **Execute Code First Migrations (runs on application start)**, then click **Save**.



Publish your changes

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

1. In the publish page, click **Publish**.
2. Try adding to-do items again and select **Done**, and they should show up in your homepage as a completed item.



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

Stream application logs

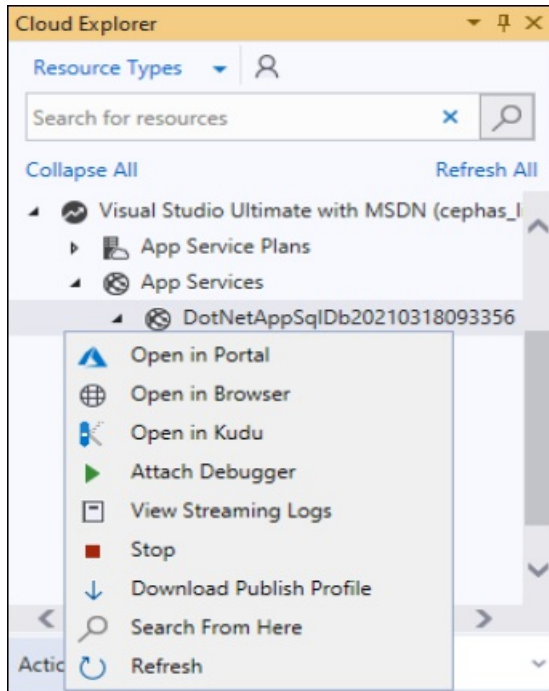
You can stream tracing messages directly from your Azure app to Visual Studio.

Open `Controllers\TodosController.cs`.

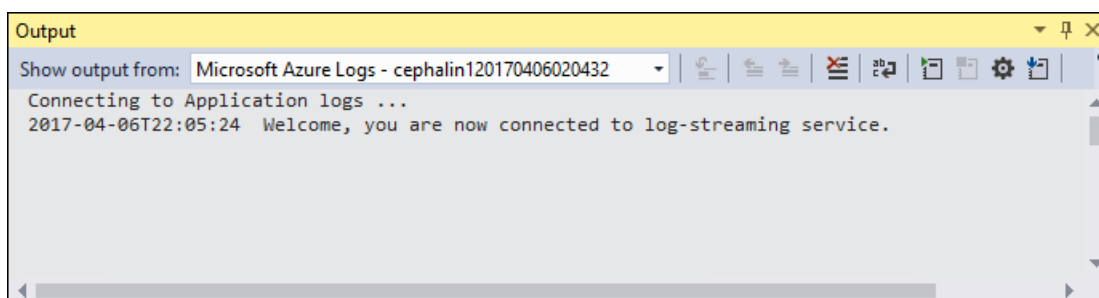
Each action starts with a `Trace.WriteLine()` method. This code is added to show you how to add trace messages to your Azure app.

Enable log streaming

1. From the **View** menu, select **Cloud Explorer**.
2. In **Cloud Explorer**, expand the Azure subscription that has your app and expand **App Service**.
3. Right-click your Azure app and select **View Streaming Logs**.



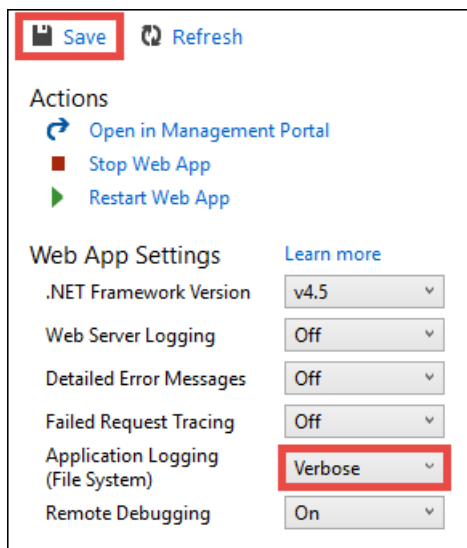
The logs are now streamed into the **Output** window.



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

Change trace levels

1. To change the trace levels to output other trace messages, go back to **Cloud Explorer**.
2. Right-click your app again and select **Open in Portal**.
3. In the portal management page for your app, from the left menu, select **App Service logs**.
4. Under **Application Logging (File System)**, select **Verbose** in **Level**. Click **Save**.



TIP

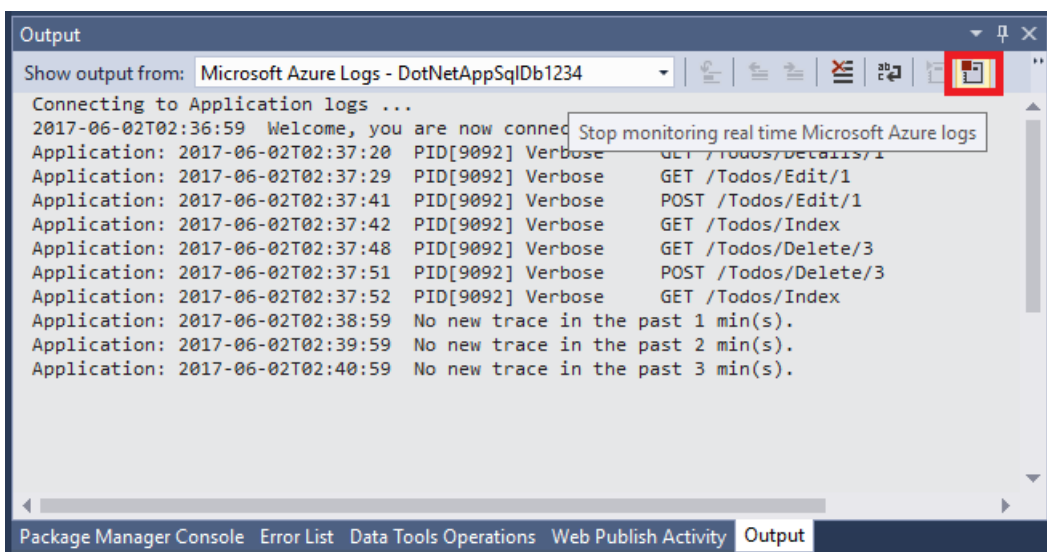
You can experiment with different trace levels to see what types of messages are displayed for each level. For example, the **Information** level includes all logs created by `Trace.TraceInformation()`, `Trace.TraceWarning()`, and `Trace.TraceError()`, but not logs created by `Trace.WriteLine()`.

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

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

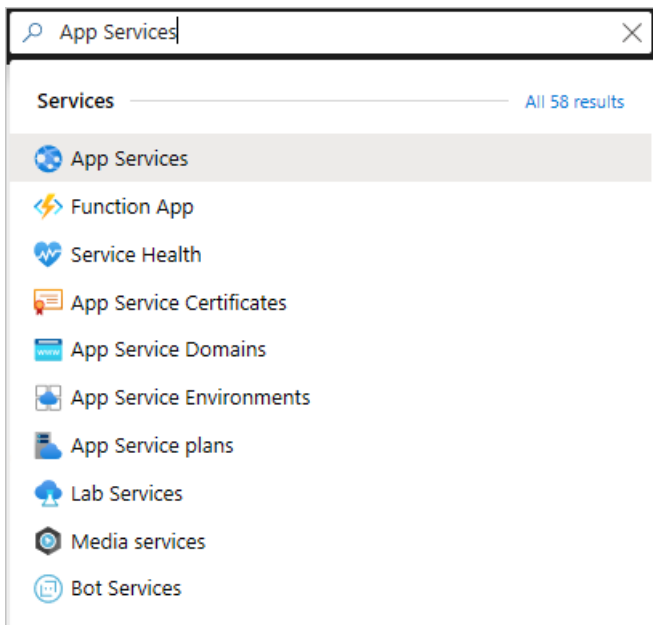
Stop log streaming

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

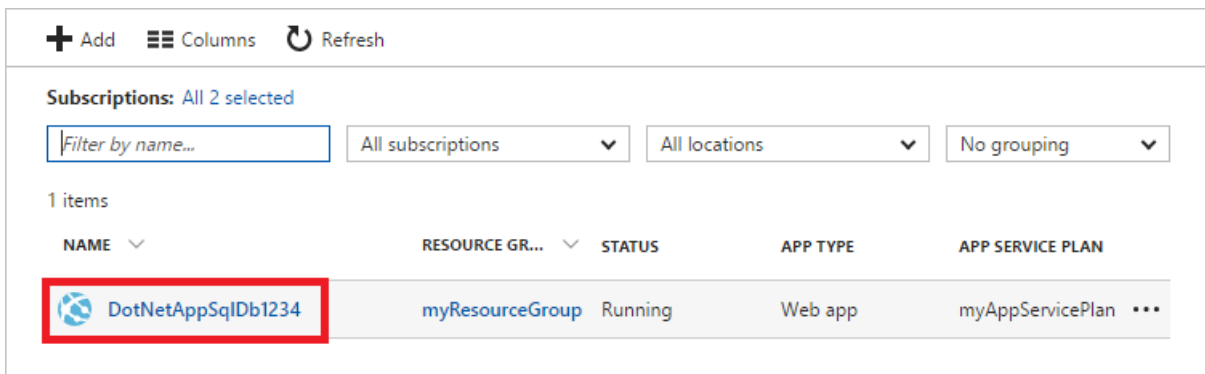


Manage your Azure app

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

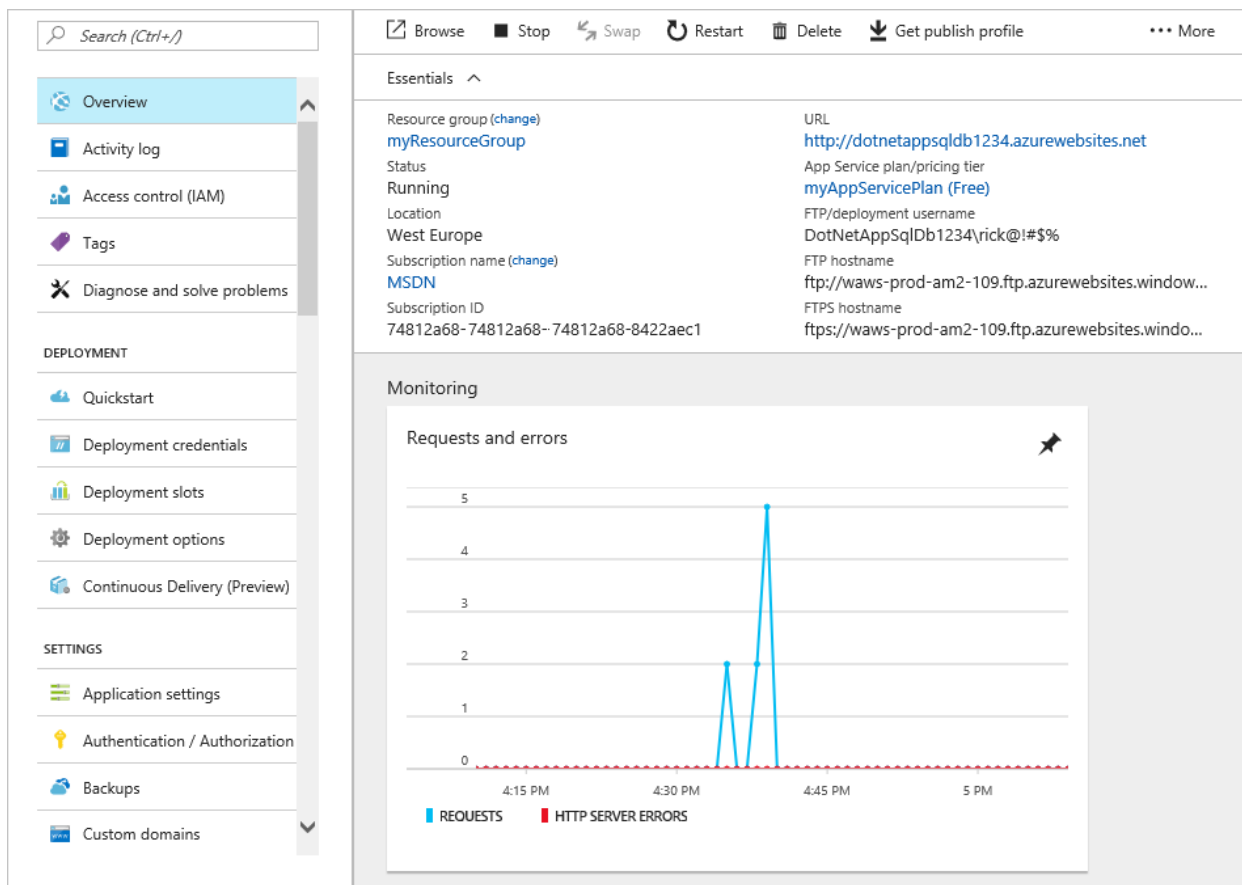


Select the name of your Azure app.



You have landed in your app's page.

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



Clean up resources

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

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

Next steps

In this tutorial, you learned how to:

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

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

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

More resources:

[Configure ASP.NET app](#)

Want to optimize and save on your cloud spending?

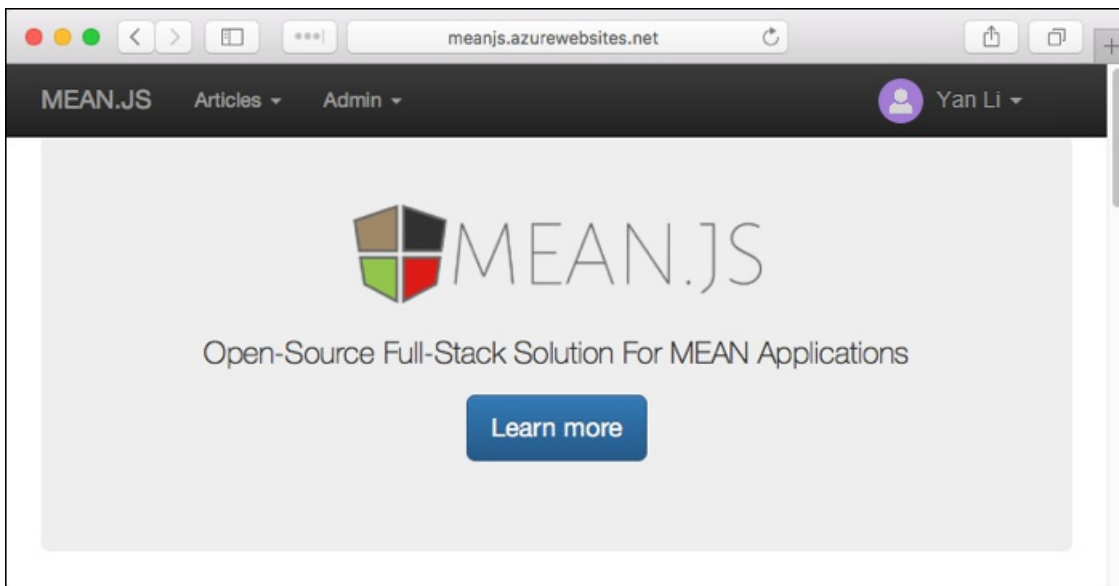
[Start analyzing costs with Cost Management](#)

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

6/25/2021 • 17 minutes to read • [Edit Online](#)

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

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



What you'll learn:

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


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

Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install Node.js and NPM](#)
- [Install Bower](#) (required by [MEAN.js](#))
- [Install Gulp.js](#) (required by [MEAN.js](#))

- [Install and run MongoDB Community Edition](#)
- Use the Bash environment in [Azure Cloud Shell](#).

 Launch Cloud Shell

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Test local MongoDB

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

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

```
mongo
```

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

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

Create local Node.js app

In this step, you set up the local Node.js project.

Clone the sample application

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

Run the following command to clone the sample repository.

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

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

Run the application

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

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

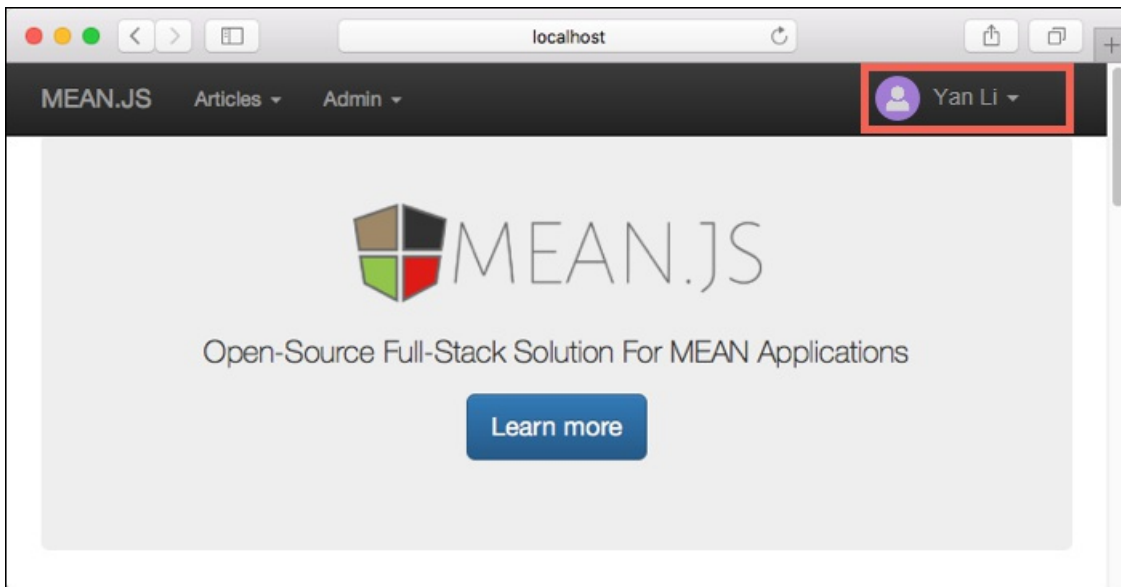
Ignore the config.domain warning. When the app is fully loaded, you see something similar to the following

message:

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

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

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



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

To stop Node.js at any time, press `Ctrl+C` in the terminal.

Create production MongoDB

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

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

Create a resource group

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

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

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

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

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

Create a Cosmos DB account

NOTE

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

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

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

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

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

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

```
{
  "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 keys list` command to retrieve the primary key.

```
az cosmosdb keys list --name <cosmosdb-name> --resource-group myResourceGroup
```

The Azure CLI shows information similar to the following example:

```
{
  "primaryMasterKey":
  "RS4CmUwzGRASJPMoc0kiEvdnKmxYRILC9BwisAYh3Hq4zBYKr0XQiSE4pqx3UchBe04QRCzUt1i7w0rOkitoJw==",
  "primaryReadOnlyMasterKey":
  "HvitsjIYz8TwRmIuPEUAALRwqgKOzJUjW22wPL2U8zoMVhGvregBkBk9LdMTxqBgDETSq7obbwZtdeFY7hElTg==",
  "secondaryMasterKey":
  "Lu9aeZTiXU4PjuuyGBbvS1N9IRG3oegIrIh95U6V0stf9bJiiIpw3IFwSUGQWSEYM3VeEyrhHJ4rn3Ci0vuFqA==",
  "secondaryReadOnlyMasterKey":
  "LpsCicpVZqHRy7qbMgrzbRKjbyCwCKPQR10QpgReA0xMcggtVxJFA94fTi0oQ7txpftTJcXkjTirQ0pT7QFrQ=="
}
```

Copy the value of `primaryMasterKey`. You need this information in the next step.

Configure the connection string in your Node.js application

In your local MEAN.js repository, in the `config/env/` folder, create a file named `local-production.js`. `.gitignore` is already 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 TLS/SSL](#).

Save your changes.

Test the application in production mode

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

```
gulp prod
```

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

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

# Windows PowerShell
$env:NODE_ENV = "production"
node server.js
```

`NODE_ENV=production` sets the environment variable that tells Node.js to run in the production environment.

`node server.js` starts the Node.js server with `server.js` in your repository root. This is how your Node.js application is loaded in Azure.

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

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

Navigate to `http://localhost:8443` in a browser. Click **Sign Up** in the top menu and create a test user. If you are successful creating a user and signing in, then your app is writing data to the Cosmos DB database in Azure.

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

Deploy app to Azure

In this step, you deploy your MongoDB-connected Node.js application to Azure App Service.

Configure a deployment user

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

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

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

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

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

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

Create an App Service plan

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

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

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

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

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

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

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

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

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

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

Create a web app

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

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `NODE|14-LTS`. To see all supported runtimes, run `az webapp list-runtimes`.

```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime "NODE|14-LTS" --deployment-local-git
# PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime "NODE|14-LTS" --deployment-local-git
```

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

```

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

```

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

NOTE

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

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

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

```

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

```

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

```

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

```

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

NOTE

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

Configure an environment variable

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

To set app settings, use the `az webapp config appsettings set` command in the Cloud Shell.

The following example configures a `MONGODB_URI` app setting in your Azure app. Replace the `<app-name>`, `<cosmosdb-name>`, and `<primary-master-key>` placeholders.

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

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

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

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

Push to Azure from Git

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

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

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

```
git push azure master
```

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

```
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 489 bytes | 0 bytes/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Updating branch 'master'.
remote: Updating submodules.
remote: Preparing deployment for commit id '6c7c716eee'.
remote: Running custom deployment command...
remote: Running deployment command...
remote: Handling node.js deployment.
.
.
.
remote: Deployment successful.
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
* [new branch]      master -> master
```

You may notice that the deployment process runs [Gulp](#) after `npm install`. App Service does not run Gulp or Grunt tasks during deployment, so this sample repository has two additional files in its root directory to enable it:

- `.deployment` - This file tells App Service to run `bash deploy.sh` as the custom deployment script.
- `deploy.sh` - The custom deployment script. If you review the file, you will see that it runs `gulp prod` after `npm install` and `bower install`.

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

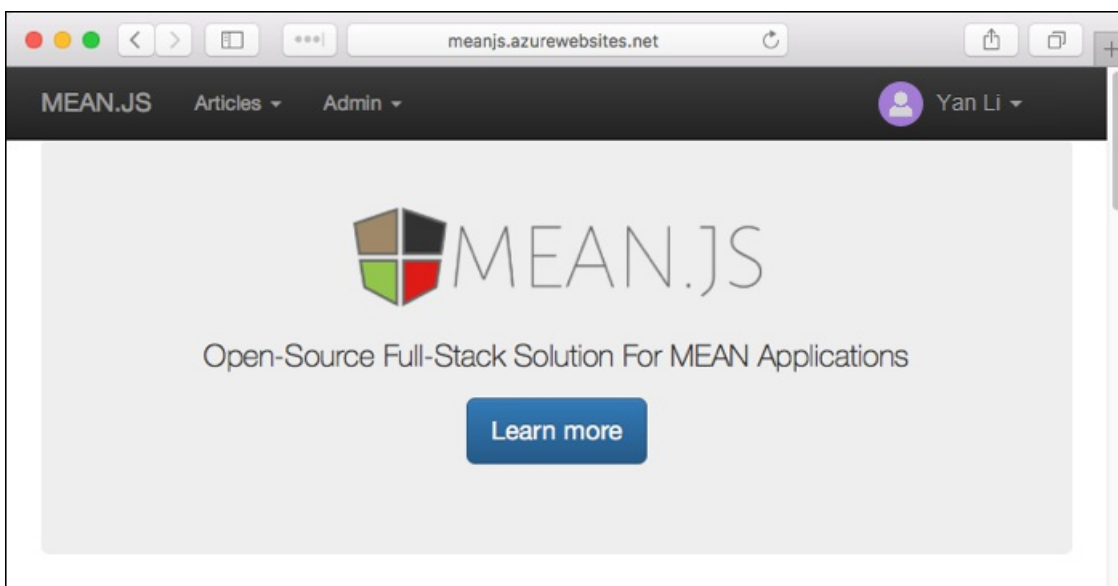
Browse to the Azure app

Browse to the deployed app using your web browser.

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

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

If you are successful and the app automatically signs in to the created user, then your MEAN.js app in Azure has connectivity to the 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

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

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

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

Update the articles code

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

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

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

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

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

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

  ...
};
```

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

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

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

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

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


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

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

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

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

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

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

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

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

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

Test your changes locally

Save all your changes.

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

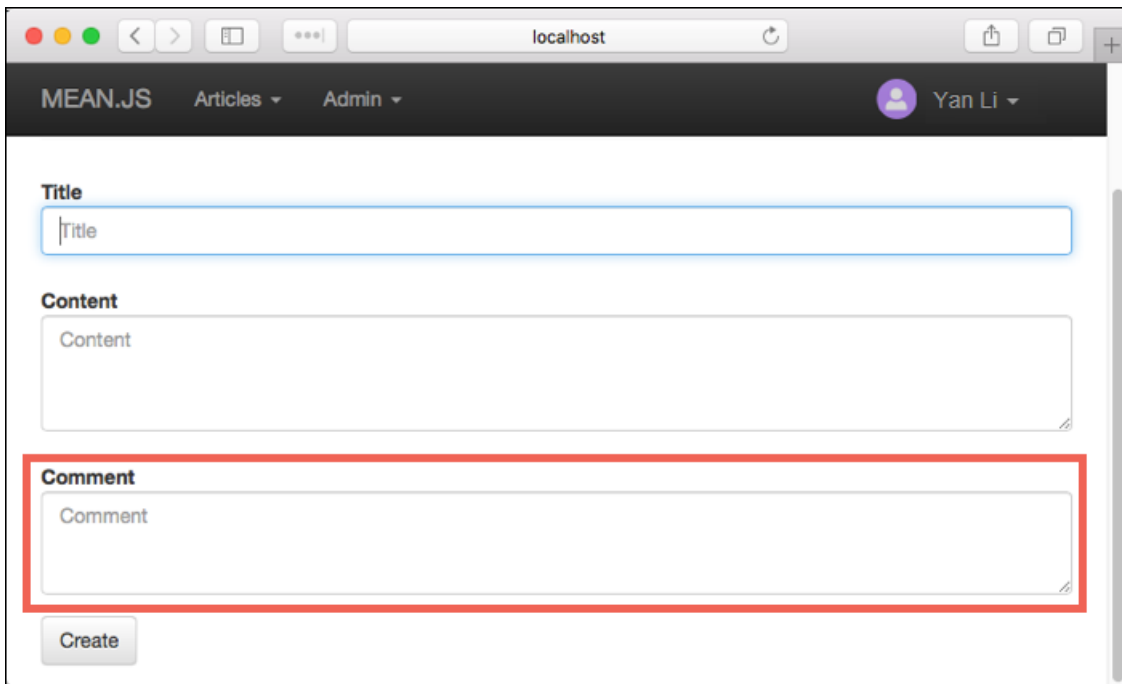
```
# Bash
gulp prod
NODE_ENV=production node server.js

# Windows PowerShell
gulp prod
$env:NODE_ENV = "production"
node server.js
```

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

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

You see the new `Comment` textbox now.



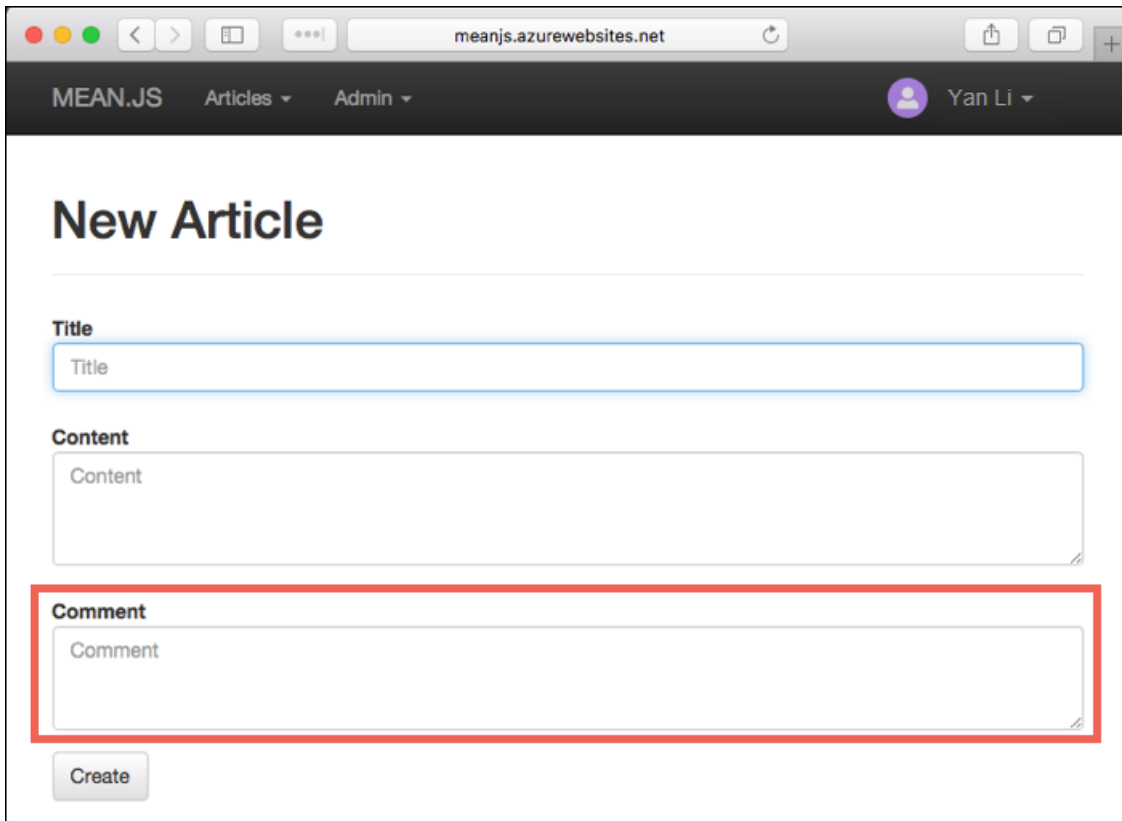
In the terminal, stop Node.js by typing `Ctrl+C`.

Publish changes to Azure

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

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

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



If you added any articles earlier, you still can see them. Existing data in your Cosmos DB is not lost. Also, your updates to the data schema and leaves your existing data intact.

Stream diagnostic logs

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

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

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

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

Stop log streaming at any time by typing `Ctrl+C`.

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --application-logging true --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

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

If you don't see console logs immediately, check again in 30 seconds.

NOTE

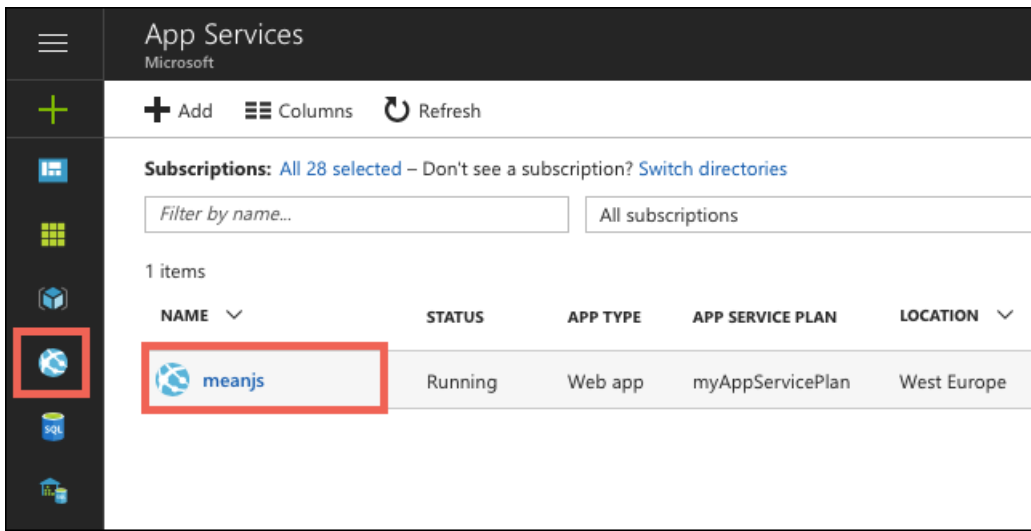
You can also inspect the log files from the browser at `https://<app-name>.scm.azurewebsites.net/api/logs/docker`.

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

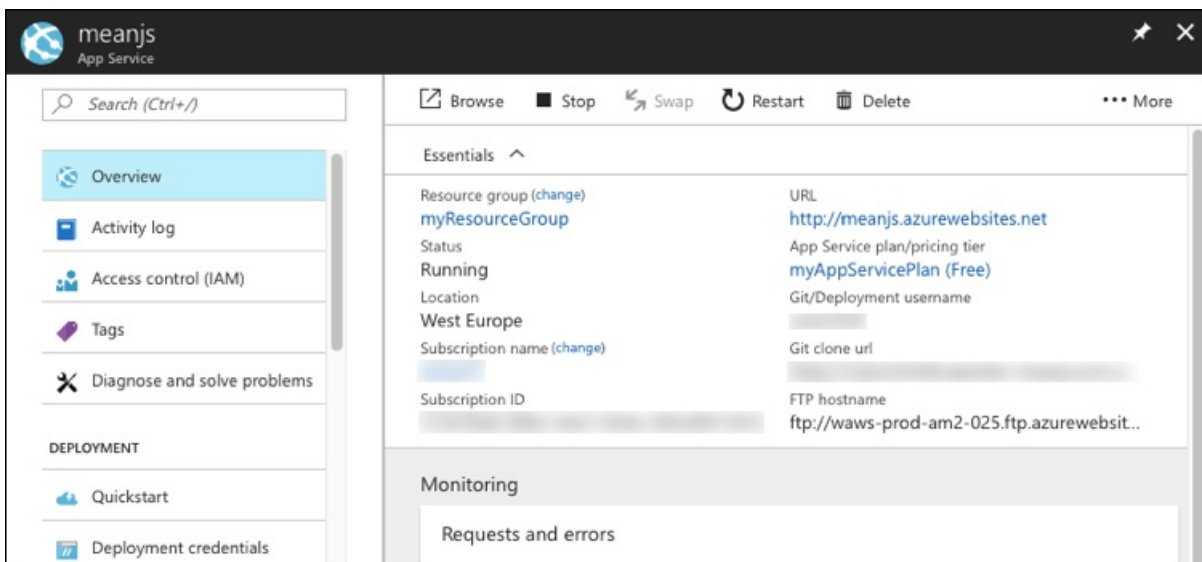
Manage your Azure app

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

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



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



Clean up resources

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

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

This command may take a minute to run.

Next steps

What you learned:

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

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

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

Or, check out other resources:

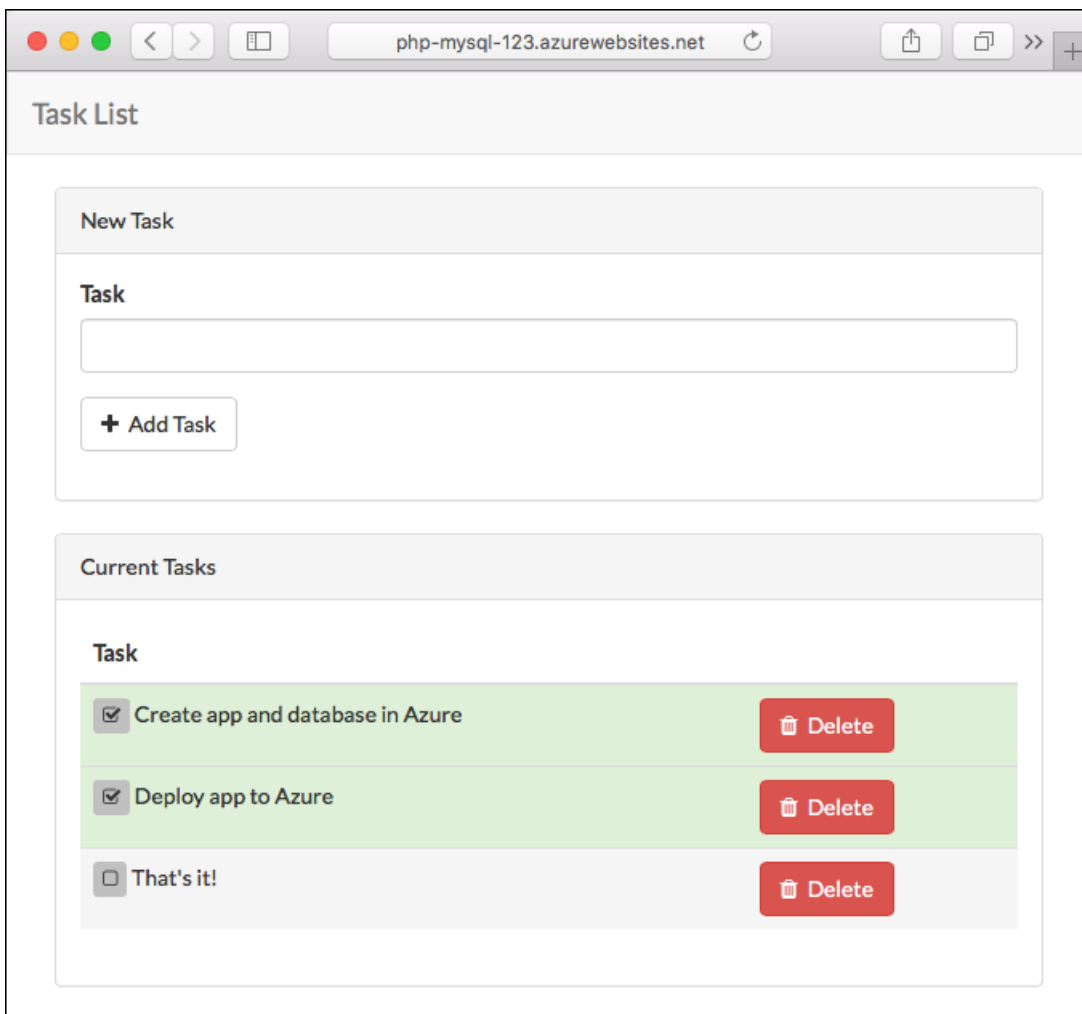
[Configure Node.js app](#)

Tutorial: Build a PHP and MySQL app in Azure App Service

4/21/2021 • 20 minutes to read • [Edit Online](#)

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

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



In this tutorial, you learn how to:


- Create a MySQL database in Azure
- Connect a PHP app to MySQL
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

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

Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install PHP 5.6.4 or above](#)
- [Install Composer](#)
- Enable the following PHP extensions Laravel needs: OpenSSL, PDO-MySQL, Mbstring, Tokenizer, XML
- [Install and start MySQL](#)
- Use the Bash environment in [Azure Cloud Shell](#).

 Launch Cloud Shell

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Prepare local MySQL

In this step, you create a database in your local MySQL server for your use in this tutorial.

Connect to local MySQL server

In a terminal window, connect to your local MySQL server. You can use this terminal window to run all the commands in this tutorial.

```
mysql -u root -p
```

If you're prompted for a password, enter the password for the `root` account. If you don't remember your root account password, see [MySQL: How to Reset the Root Password](#).

If your command runs successfully, then your MySQL server is running. If not, make sure that your local MySQL server is started by following the [MySQL post-installation steps](#).

Create a database locally

At the `mysql` prompt, create a database.

```
CREATE DATABASE sampledb;
```

Exit your server connection by typing `quit`.

```
quit
```

Create a PHP app locally

In this step, you get a Laravel sample application, configure its database connection, and run it locally.

Clone the sample

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

Run the following command to clone the sample repository.

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

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

```
cd laravel-tasks
composer install
```

Configure MySQL connection

In the repository root, create a file named `.env`. Copy the following variables into the `.env` file. Replace the `<root_password>` placeholder with the MySQL root user's password.

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=

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
```

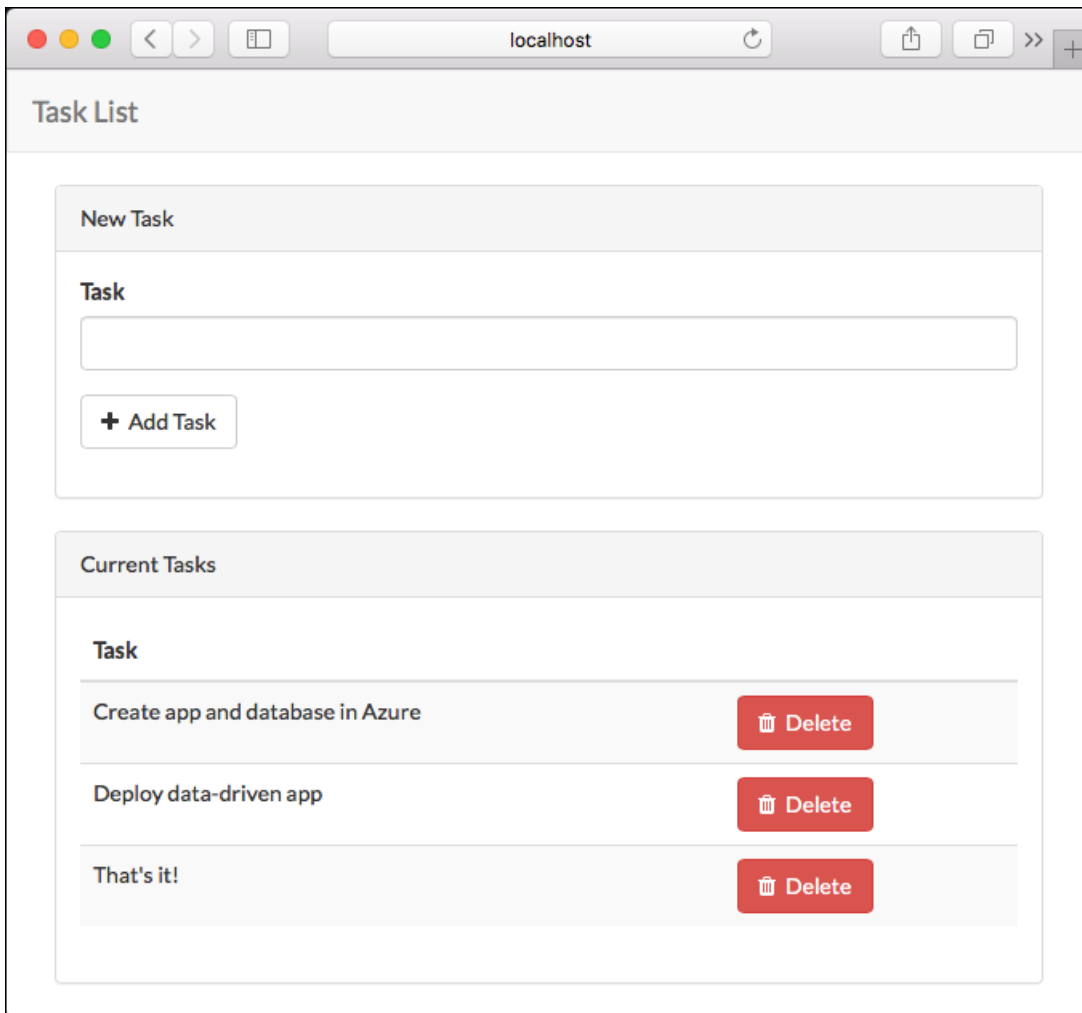
Generate a new Laravel application key.

```
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.



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

Create MySQL in Azure

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

Create a resource group

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

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

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

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

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

Create a MySQL server

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

In the following command, substitute a unique server name for the `<mysql-server-name>` placeholder, a user name for the `<admin-user>`, and a password for the `<admin-password>` placeholder. The server name is used

as part of your MySQL endpoint (`https://<mysql-server-name>.mysql.database.azure.com`), so the name needs to be unique across all servers in Azure. For details on selecting MySQL DB SKU, see [Create an Azure Database for MySQL server](#).

```
az mysql server create --resource-group myResourceGroup --name <mysql-server-name> --location "West Europe"
--admin-user <admin-user> --admin-password <admin-password> --sku-name B_Gen5_1
```

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

```
{
  "administratorLogin": "<admin-user>",
  "administratorLoginPassword": null,
  "fullyQualifiedDomainName": "<mysql-server-name>.mysql.database.azure.com",
  "id": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/<mysql-server-name>",
  "location": "westeurope",
  "name": "<mysql-server-name>",
  "resourceGroup": "myResourceGroup",
  ...
  - < Output has been truncated for readability >
}
```

Configure server firewall

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

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

TIP

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

In the Cloud Shell, run the command again to allow access from your local computer by replacing `<your-ip-address>` with [your local IPv4 IP address](#).

```
az mysql server firewall-rule create --name AllowLocalClient --server <mysql-server-name> --resource-group
myResourceGroup --start-ip-address=<your-ip-address> --end-ip-address=<your-ip-address>
```

Connect to production MySQL server locally

In the local terminal window, connect to the MySQL server in Azure. Use the value you specified previously for `<admin-user>` and `<mysql-server-name>`. When prompted for a password, use the password you specified when you created the database in Azure.

```
mysql -u <admin-user>@<mysql-server-name> -h <mysql-server-name>.mysql.database.azure.com -P 3306 -p
```

Create a production database

At the `mysql` prompt, create a database.

```
CREATE DATABASE sampledb;
```

Create a user with permissions

Create a database user called *phpappuser* and give it all privileges in the `sampledb` database. For simplicity of the tutorial, use *MySQLAzure2017* as the password.

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

Exit the server connection by typing `quit`.

```
quit
```

Connect app to Azure MySQL

In this step, you connect the PHP application to the MySQL database you created in Azure Database for MySQL.

Configure the database connection

In the repository root, create an *.env.production* file and copy the following variables into it. Replace the placeholder `<mysql-server-name>` in both *DB_HOST* and *DB_USERNAME*.

```
APP_ENV=production  
APP_DEBUG=true  
APP_KEY=  
  
DB_CONNECTION=mysql  
DB_HOST=<mysql-server-name>.mysql.database.azure.com  
DB_DATABASE=sampledb  
DB_USERNAME=phpappuser@<mysql-server-name>  
DB_PASSWORD=MySQLAzure2017  
MYSQL_SSL=true
```

Save the changes.

TIP

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

Configure TLS/SSL certificate

By default, Azure Database for MySQL enforces TLS connections from clients. To connect to your MySQL database in Azure, you must use the [.pem certificate supplied by Azure Database for MySQL](#).

Open *config/database.php* and add the `sslmode` and `options` parameters to `connections.mysql`, as shown in the following code.

```
'mysql' => [
    ...
    'sslmode' => env('DB_SSLMODE', 'prefer'),
    'options' => (env('MYSQL_SSL')) ? [
        PDO::MYSQL_ATTR_SSL_KEY => '/ssl/BaltimoreCyberTrustRoot.crt.pem',
    ] : []
],
```

```
'mysql' => [
    ...
    'sslmode' => env('DB_SSLMODE', 'prefer'),
    'options' => (env('MYSQL_SSL') && extension_loaded('pdo_mysql')) ? [
        PDO::MYSQL_ATTR_SSL_KEY => '/ssl/BaltimoreCyberTrustRoot.crt.pem',
    ] : []
],
```

The certificate `BaltimoreCyberTrustRoot.crt.pem` is provided in the repository for convenience in this tutorial.

Test the application locally

Run Laravel database migrations with `.env.production` as the environment file to create the tables in your MySQL database in Azure Database for MySQL. Remember that `.env.production` has the connection information to your MySQL database in Azure.

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

`.env.production` doesn't have a valid application key yet. Generate a new one for it in the terminal.

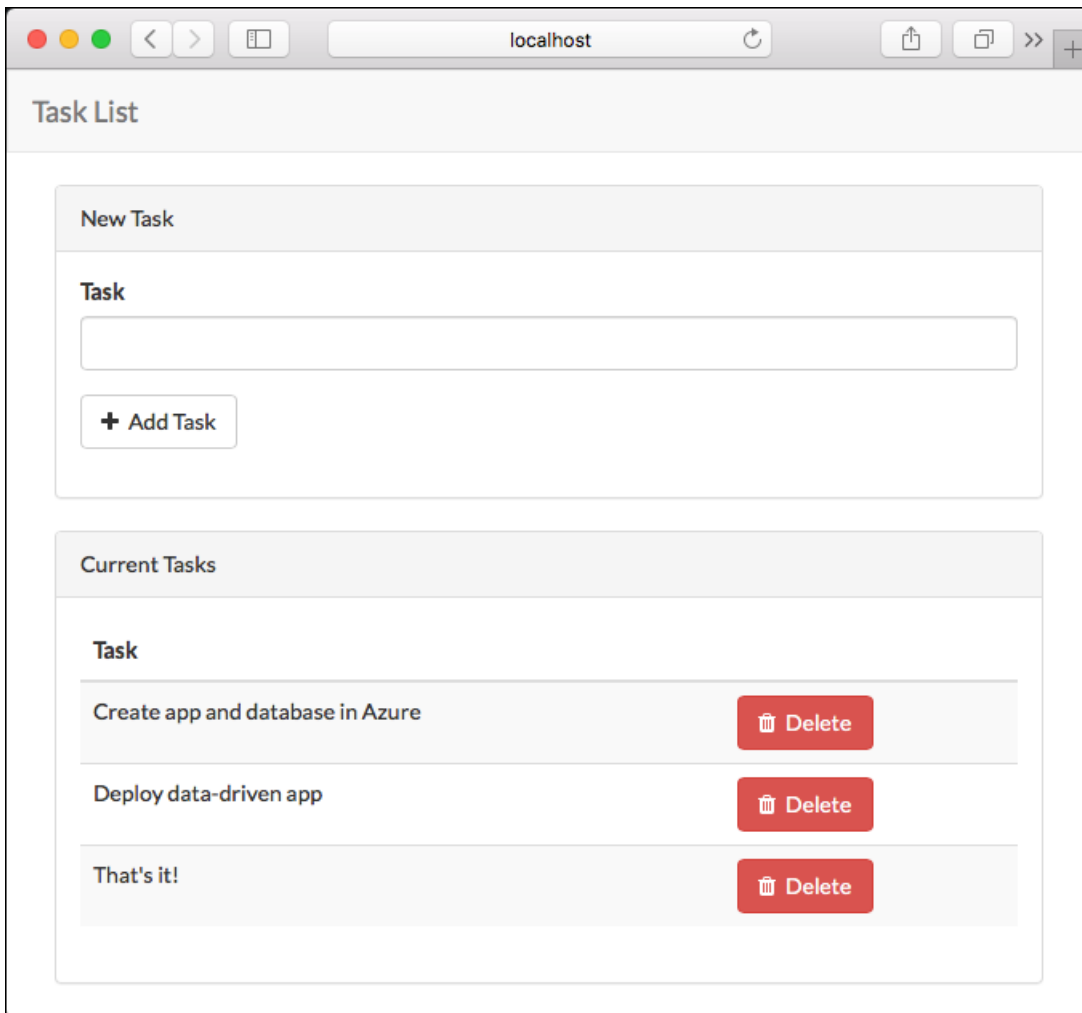
```
php artisan key:generate --env=production --force
```

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

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

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

Add a few tasks in the page.



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

Commit your changes

Run the following Git commands to commit your changes:

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

Your app is ready to be deployed.

Deploy to Azure

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

Configure a deployment user

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

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

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

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

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

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

Create an App Service plan

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

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

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

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

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

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

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

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

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

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

Create a web app

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

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `PHP|7.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
# PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
```

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

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

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

NOTE

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

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

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `PHP|7.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
# PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
```

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

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

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

NOTE

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

Configure database settings

In App Service, you set environment variables as *app settings* by using the `az webapp config appsettings set` command.

The following command configures the app settings `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`. Replace the placeholders `<app-name>` and `<mysql-server-name>`.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings DB_HOST="
<mysql-server-name>.mysql.database.azure.com" DB_DATABASE="sampledb" DB_USERNAME="phpappuser@<mysql-server-
name>" DB_PASSWORD="MySQLAzure2017" MYSQL_SSL="true"
```

You can use the PHP `getenv` method to access the settings. the Laravel code uses an `env` wrapper over the PHP `getenv`. For example, the MySQL configuration in `config/database.php` looks like the following code:


```
'mysql' => [
  'driver'    => 'mysql',
  'host'     => env('DB_HOST', 'localhost'),
  'database' => env('DB_DATABASE', 'forge'),
  'username' => env('DB_USERNAME', 'forge'),
  'password' => env('DB_PASSWORD', ''),
  ...
],
```

Configure Laravel environment variables

Laravel needs an application key in App Service. You can configure it with app settings.

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

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

In the Cloud Shell, set the application key in the App Service app by using the `az webapp config appsettings set` command. Replace the placeholders `<app-name>` and `<outputofphpartisankey:generate>`.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings APP_KEY="
<output_of_php_artisan_key:generate>" APP_DEBUG="true"
```

`APP_DEBUG="true"` tells Laravel to return debugging information when the deployed app encounters errors.

When running a production application, set it to `false`, which is more secure.

Set the virtual application path

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

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

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

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

[Laravel application lifecycle](#) begins in the `public` directory instead of the application's root directory. The default PHP Docker image for App Service uses Apache, and it doesn't let you customize the `DocumentRoot` for Laravel. However, you can use `.htaccess` to rewrite all requests to point to `/public` instead of the root directory. In the repository root, an `.htaccess` is added already for this purpose. With it, your Laravel application is ready to be deployed.

For more information, see [Change site root](#).

Push to Azure from Git

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

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

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

```
git push azure master
```

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

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

NOTE

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

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

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

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

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

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

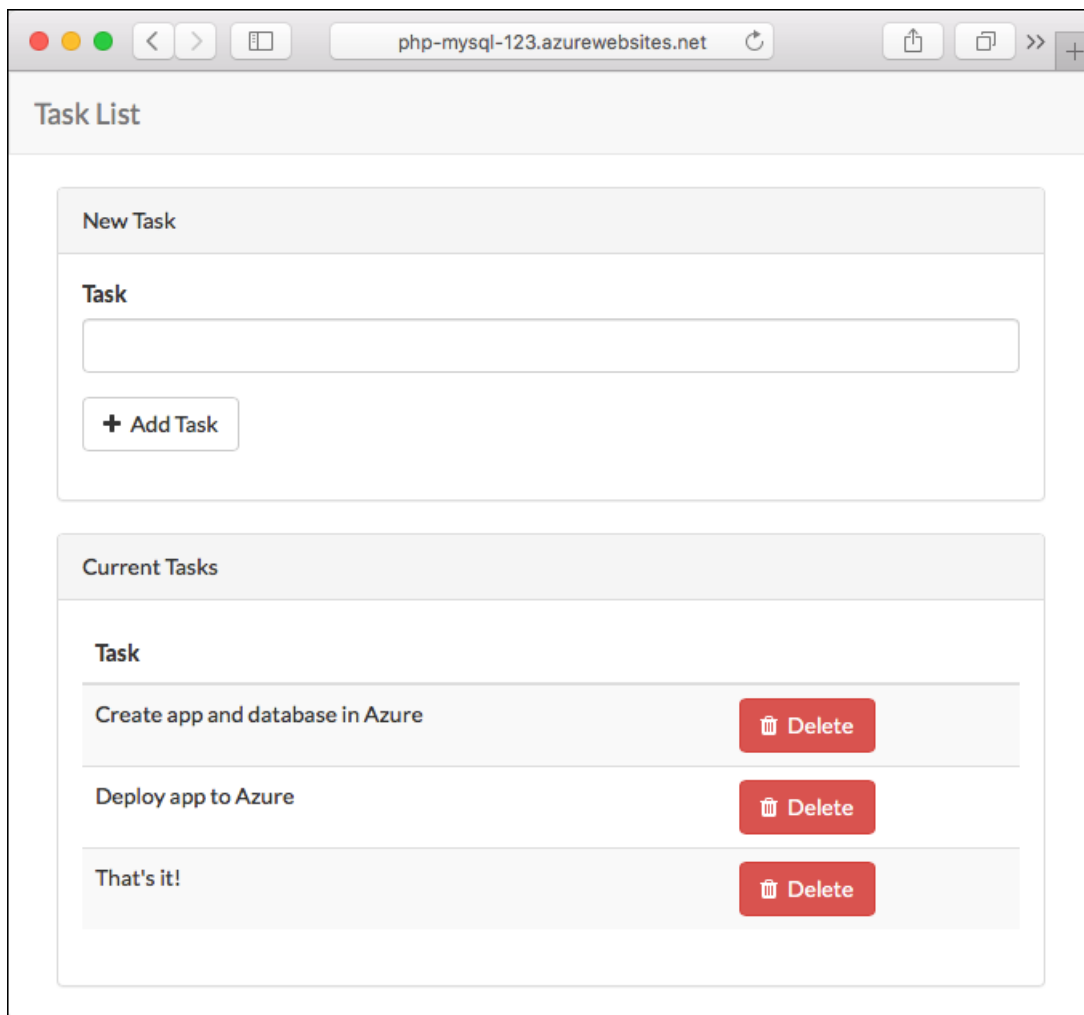
```
git push azure master
```

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

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

Browse to the Azure app

Browse to `http://<app-name>.azurewebsites.net` and add a few tasks to the list.



Congratulations, you're running a data-driven PHP app in Azure App Service.

Update model locally and redeploy

In this step, you make a simple change to the `task` data model and the webapp, and then publish the update to Azure.

For the tasks scenario, you modify the application so that you can mark a task as complete.

Add a column

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. Find the `<tr>` opening tag and replace it with:

```
<tr class="{{ $task->complete ? 'success' : 'active' }}" >
```

The preceding code changes the row color depending on whether the task is complete.

In the next line, you have the following code:

```
<td class="table-text"><div>{{ $task->name }}</div></td>
```

Replace the entire line with the following code:

```
<td>
  <form action="{{ url('task/'. $task->id) }}" method="POST">
    {{ csrf_field() }}

    <button type="submit" class="btn btn-xs">
      <i class="fa {{$task->complete ? 'fa-check-square-o' : 'fa-square-o'}}"></i>
    </button>
    {{ $task->name }}
  </form>
</td>
```

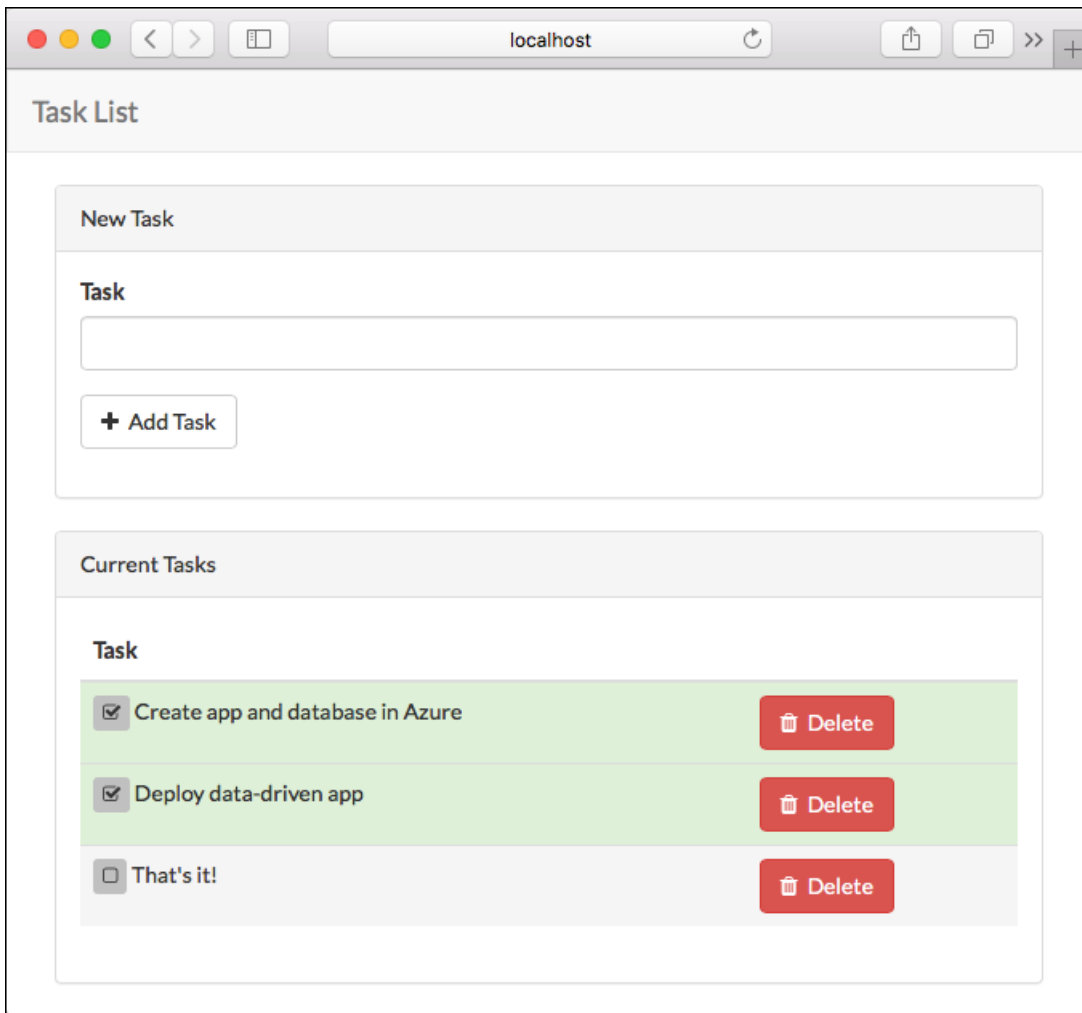
The preceding code adds the submit button that references the route that you defined earlier.

Test the changes locally

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

```
php artisan serve
```

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



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

Publish changes to Azure

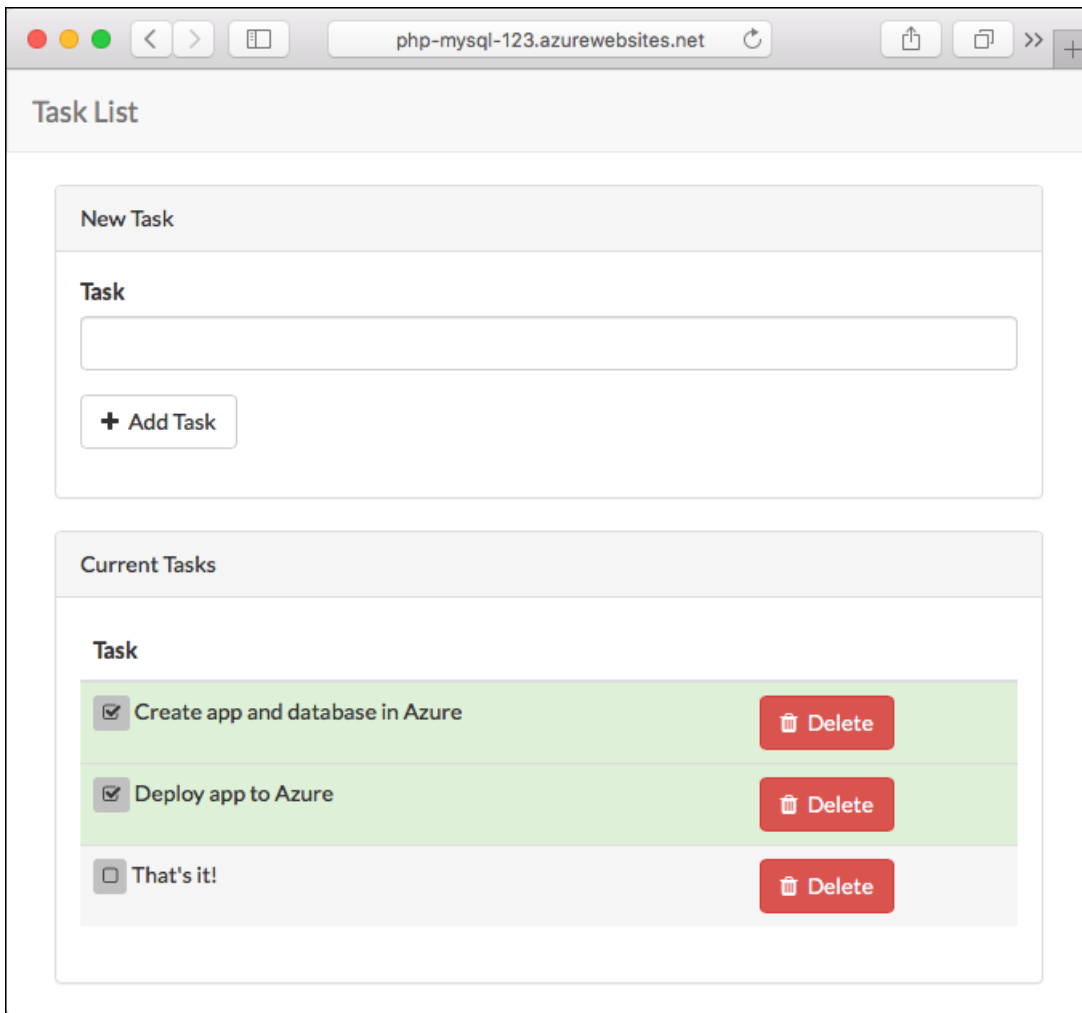
In the local terminal window, run Laravel database migrations with the production connection string to make the change in the Azure database.

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

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

```
git add .
git commit -m "added complete checkbox"
git push azure main
```

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



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

Stream diagnostic logs

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

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

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

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

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

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --application-logging true --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

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

If you don't see console logs immediately, check again in 30 seconds.

NOTE

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

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

TIP

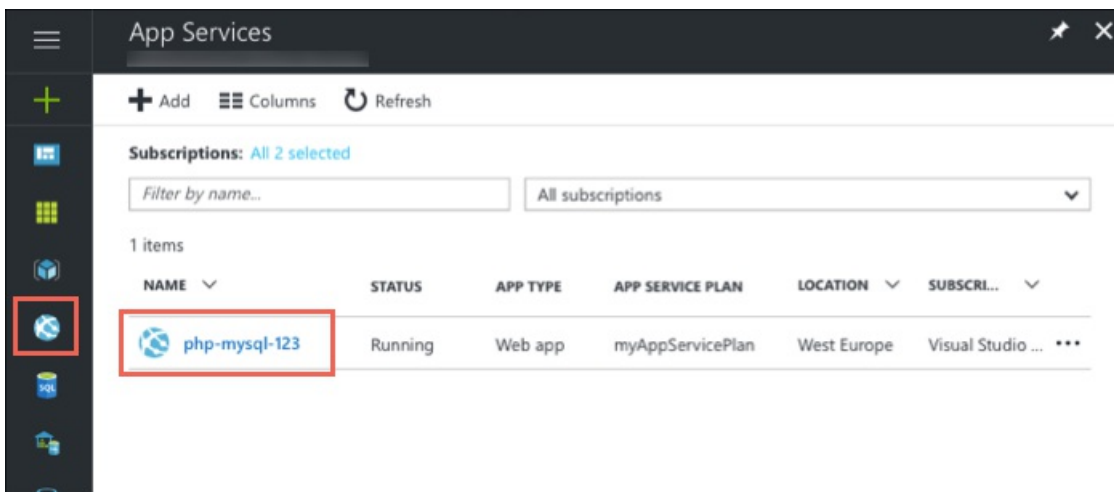
A PHP application can use the standard `error_log()` to output to the console. The sample application uses this approach in `app/Http/routes.php`.

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

Manage the Azure app

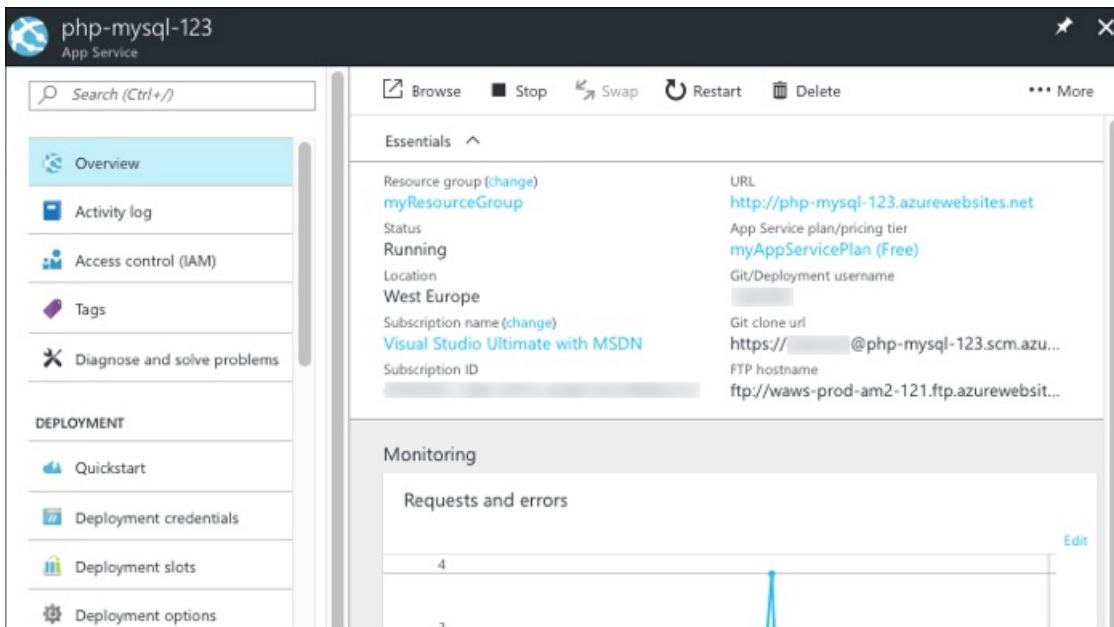
Go to the [Azure portal](#) to manage the app you created.

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



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

The left menu provides pages for configuring your app.



Clean up resources

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

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

This command may take a minute to run.

Next steps

In this tutorial, you learned how to:

- Create a MySQL database in Azure
- Connect a PHP app to MySQL
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

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

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

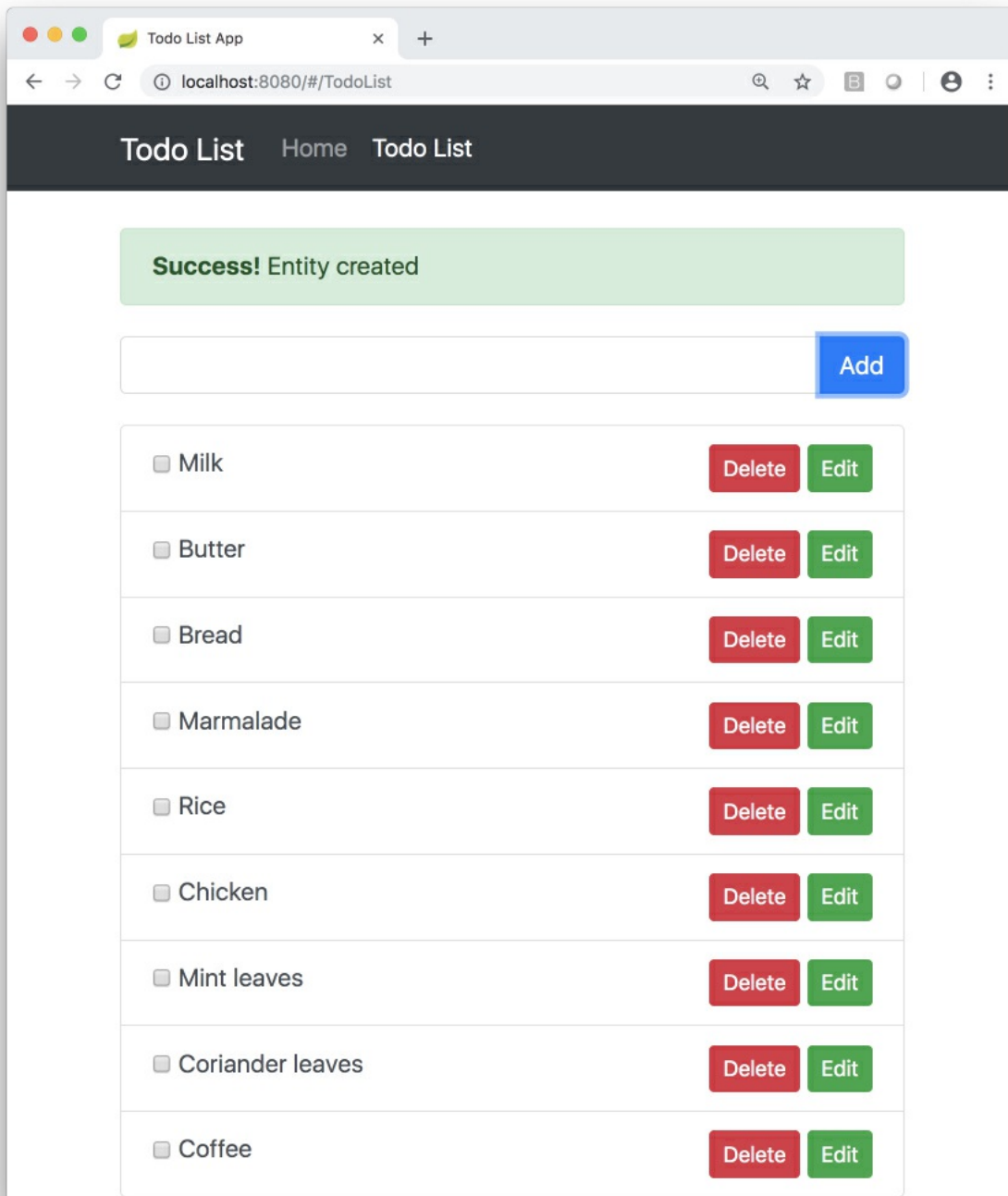
Or, check out other resources:

[Configure PHP app](#)

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

6/28/2021 • 6 minutes to read • [Edit Online](#)

This tutorial walks you through the process of building, configuring, deploying, and scaling Java web apps on Azure. When you are finished, you will have a [Spring Boot](#) application storing data in [Azure Cosmos DB](#) running on [Azure App Service on Linux](#).



In this tutorial, you learn how to:

- Create a Cosmos DB database.
- Connect a sample app to the database and test it locally

- Deploy the sample app to Azure
- Stream diagnostic logs from App Service
- Add additional instances to scale out the sample app

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

Prerequisites

- [Azure CLI](#), installed on your own computer.
- [Git](#)
- [Java JDK](#)
- [Maven](#)

Clone the sample TODO app and prepare the repo

This tutorial uses a sample TODO list app with a web UI that calls a Spring REST API backed by [Spring Data Azure Cosmos DB](#). The code for the app is available [on GitHub](#). To learn more about writing Java apps using Spring and Cosmos DB, see the [Spring Boot Starter with the Azure Cosmos DB SQL API tutorial](#) and the [Spring Data Azure Cosmos DB quick start](#).

Run the following commands in your terminal to clone the sample repo and set up the sample app environment.

```
git clone --recurse-submodules https://github.com/Azure-Samples/e2e-java-experience-in-app-service-linux-part-2.git
cd e2e-java-experience-in-app-service-linux-part-2
yes | cp -rf .prep/* .
```

Create an Azure Cosmos DB

Follow these steps to create an Azure Cosmos DB database in your subscription. The TODO list app will connect to this database and store its data when running, persisting the application state no matter where you run the application.

1. Login to your Azure CLI, and optionally set your subscription if you have more than one connected to your login credentials.

```
az login
az account set -s <your-subscription-id>
```

2. Create an Azure Resource Group, noting the resource group name.

```
az group create -n <your-azure-group-name> \
-l <your-resource-group-region>
```

3. Create Azure Cosmos DB with the `GlobalDocumentDB` kind. The name of Cosmos DB must use only lower case letters. Note down the `documentEndpoint` field in the response from the command.

```
az cosmosdb create --kind GlobalDocumentDB \
-g <your-azure-group-name> \
-n <your-azure-COSMOS-DB-name-in-lower-case-letters>
```

4. Get your Azure Cosmos DB key to connect to the app. Keep the `primaryMasterKey`, `documentEndpoint` nearby as you'll need them in the next step.

```
az cosmosdb keys list -g <your-azure-group-name> -n <your-azure-COSMOSDB-name>
```

Configure the TODO app properties

Open a terminal on your computer. Copy the sample script file in the cloned repo so you can customize it for your Cosmos DB database you just created.

```
cd initial/spring-todo-app
cp set-env-variables-template.sh .scripts/set-env-variables.sh
```

Edit `.scripts/set-env-variables.sh` in your favorite editor and supply Azure Cosmos DB connection info. For the App Service Linux configuration, use the same region as before (`your-resource-group-region`) and resource group (`your-azure-group-name`) used when creating the Cosmos DB database. Choose a `WEBAPP_NAME` that is unique since it cannot duplicate any web app name in any Azure deployment.

```
export COSMOSDB_URI=<put-your-COSMOS-DB-documentEndpoint-URI-here>
export COSMOSDB_KEY=<put-your-COSMOS-DB-primaryMasterKey-here>
export COSMOSDB_DBNAME=<put-your-COSMOS-DB-name-here>

# App Service Linux Configuration
export RESOURCEGROUP_NAME=<put-your-resource-group-name-here>
export WEBAPP_NAME=<put-your-Webapp-name-here>
export REGION=<put-your-REGION-here>
```

Then run the script:

```
source .scripts/set-env-variables.sh
```

These environment variables are used in `application.properties` in the TODO list app. The fields in the properties file set up a default repository configuration for Spring Data:

```
azure.cosmosdb.uri=${COSMOSDB_URI}
azure.cosmosdb.key=${COSMOSDB_KEY}
azure.cosmosdb.database=${COSMOSDB_DBNAME}
```

```
@Repository
public interface TodoItemRepository extends DocumentDbRepository<TodoItem, String> {
}
```

Then the sample app uses the `@Document` annotation imported from

`com.microsoft.azure.spring.data.cosmosdb.core.mapping.Document` to set up an entity type to be stored and managed by Cosmos DB:

```
@Document
public class TodoItem {
    private String id;
    private String description;
    private String owner;
    private boolean finished;
```

Run the sample app

Use Maven to run the sample.

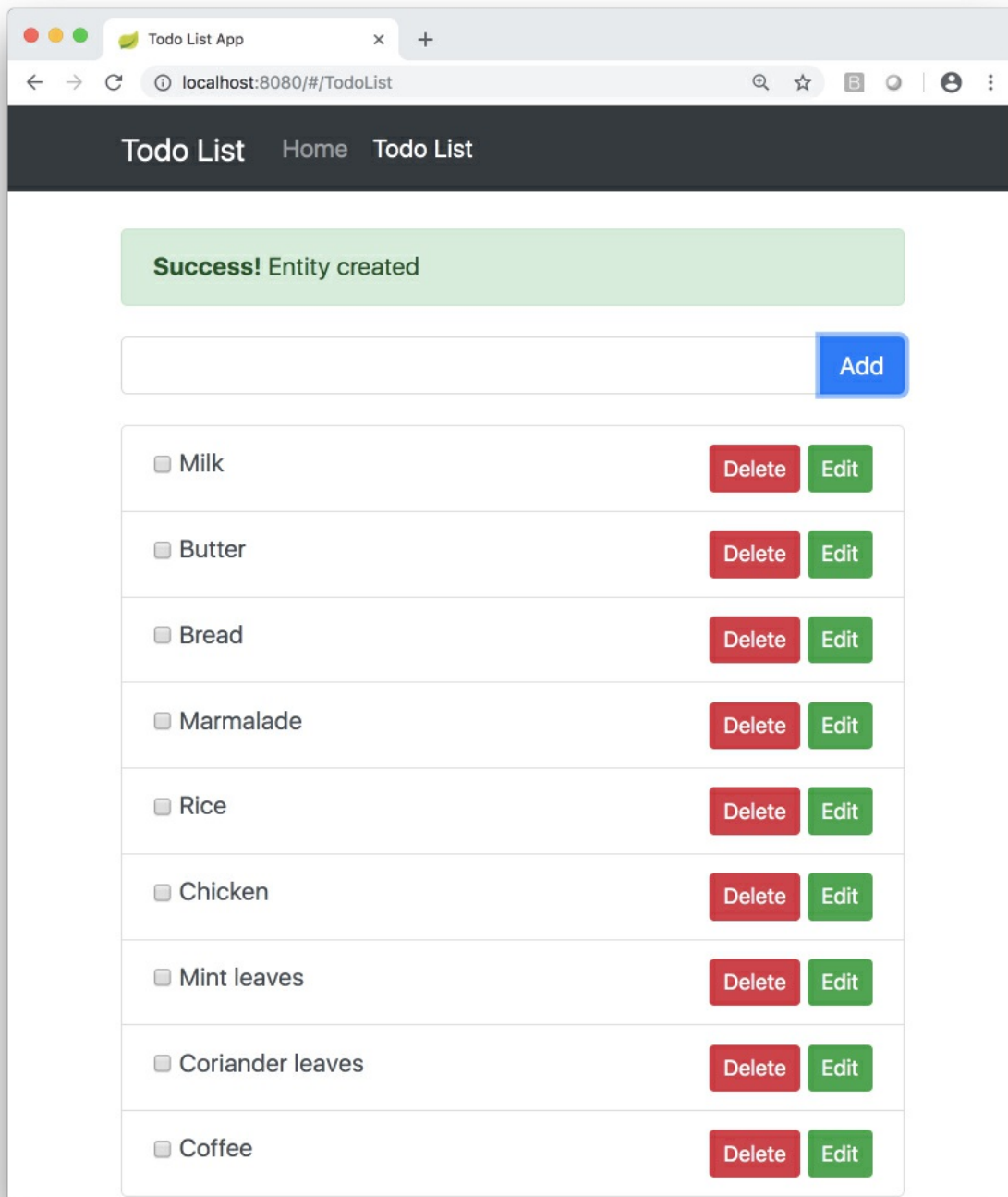
```
mvn package spring-boot:run
```

The output should look like the following.

```
bash-3.2$ mvn package spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building spring-todo-app 2.0-SNAPSHOT
[INFO] -----
[INFO]

[INFO] SimpleUrlHandlerMapping - Mapped URL path [/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[INFO] SimpleUrlHandlerMapping - Mapped URL path [/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[INFO] WelcomePageHandlerMapping - Adding welcome page: class path resource [static/index.html]
2018-10-28 15:04:32.101 INFO 7673 --- [          main] c.m.azure.documentdb.DocumentClient      :
Initializing DocumentClient with serviceEndpoint [https://sample-cosmos-db-westus.documents.azure.com:443/],
ConnectionPolicy [ConnectionPolicy [requestTimeout=60, mediaRequestTimeout=300, connectionMode=Gateway,
mediaReadMode=Buffered, maxPoolSize=800, idleConnectionTimeout=60, userAgentSuffix=;spring-
data/2.0.6;098063be661ab767976bd5a2ec350e978faba99348207e8627375e8033277cb2,
retryOptions=com.microsoft.azure.documentdb.RetryOptions@6b9fb84d, enableEndpointDiscovery=true,
preferredLocations=null]], ConsistencyLevel [null]
[INFO] AnnotationMBeanExporter - Registering beans for JMX exposure on startup
[INFO] TomcatWebServer - Tomcat started on port(s): 8080 (http) with context path ''
[INFO] TodoApplication - Started TodoApplication in 45.573 seconds (JVM running for 76.534)
```

You can access Spring TODO App locally using this link once the app is started: `http://localhost:8080/`.



If you see exceptions instead of the "Started TodoApplication" message, check that the `bash` script in the previous step exported the environment variables properly and that the values are correct for the Azure Cosmos DB database you created.

Configure Azure deployment

Open the `pom.xml` file in the `initial/spring-boot-todo` directory and add the following [Azure Web App Plugin for Maven](#) configuration.

```

<plugins>

<!--*****-->
<!-- Deploy to Java SE in App Service Linux -->
<!--*****-->

<plugin>
  <groupId>com.microsoft.azure</groupId>
  <artifactId>azure-webapp-maven-plugin</artifactId>
  <version>2.0.0</version>
  <configuration>
    <schemaVersion>v2</schemaVersion>

    <!-- Web App information -->
    <resourceGroup>${RESOURCEGROUP_NAME}</resourceGroup>
    <appName>${WEBAPP_NAME}</appName>
    <region>${REGION}</region>
    <pricingTier>P1v2</pricingTier>
    <!-- Java Runtime Stack for Web App on Linux-->
    <runtime>
      <os>linux</os>
      <javaVersion>Java 8</javaVersion>
      <webContainer>Java SE</webContainer>
    </runtime>
    <deployment>
      <resources>
        <resource>
          <directory>${project.basedir}/target</directory>
          <includes>
            <include>*.jar</include>
          </includes>
        </resource>
      </resources>
    </deployment>

    <appSettings>
      <property>
        <name>COSMOSDB_URI</name>
        <value>${COSMOSDB_URI}</value>
      </property>
      <property>
        <name>COSMOSDB_KEY</name>
        <value>${COSMOSDB_KEY}</value>
      </property>
      <property>
        <name>COSMOSDB_DBNAME</name>
        <value>${COSMOSDB_DBNAME}</value>
      </property>
      <property>
        <name>JAVA_OPTS</name>
        <value>-Dserver.port=80</value>
      </property>
    </appSettings>

  </configuration>
</plugin>
...
</plugins>

```

Deploy to App Service on Linux

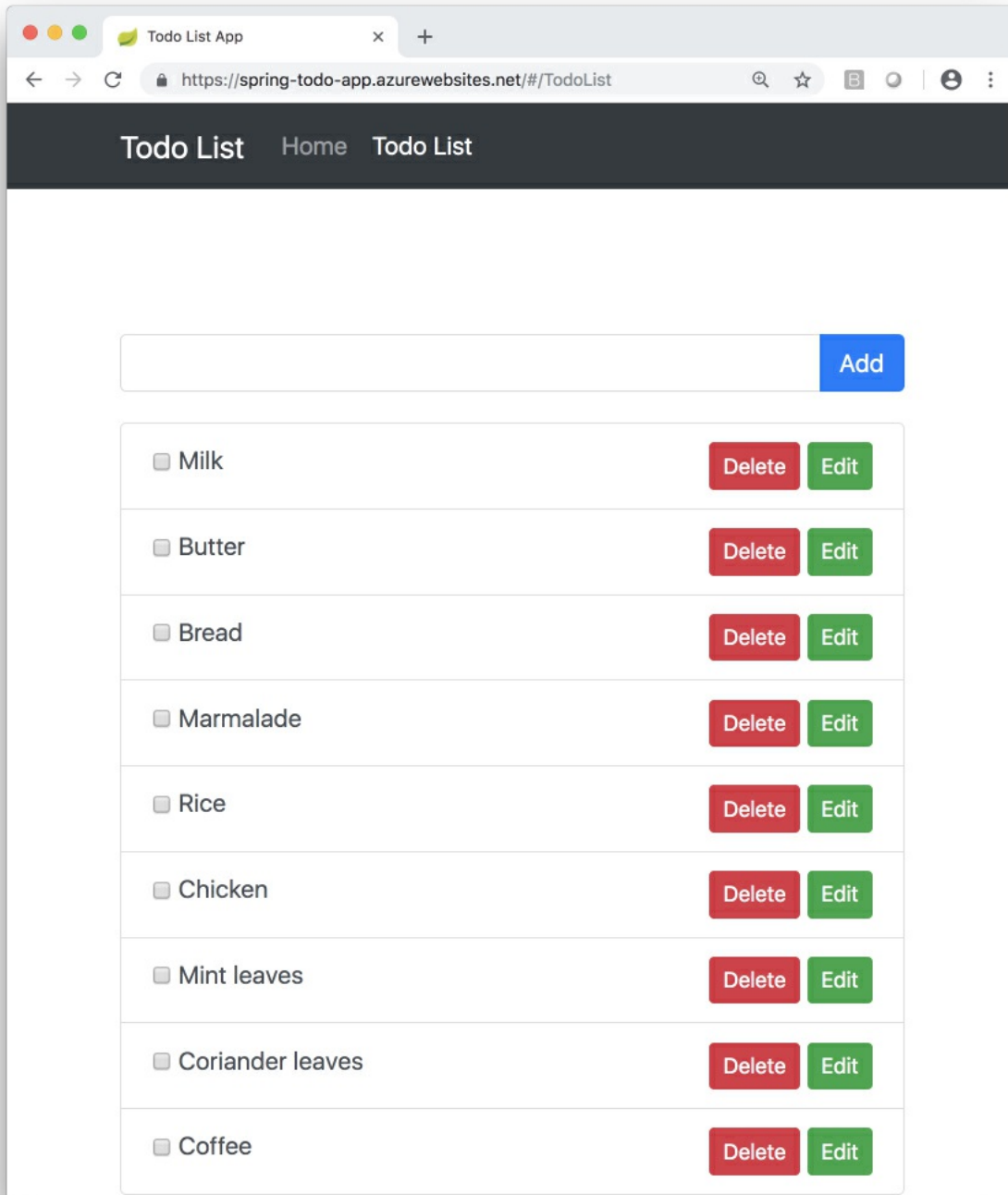
Use the `mvn azure-webapp:deploy` Maven goal to deploy the TODO app to Azure App Service on Linux.

```
# Deploy
bash-3.2$ mvn azure-webapp:deploy
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building spring-todo-app 2.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- azure-webapp-maven-plugin:2.0.0:deploy (default-cli) @ spring-todo-app ---
Auth Type: AZURE_CLI
Default subscription: xxxxxxxxx
Username: xxxxxxxxx
[INFO] Subscription: xxxxxxxxx
[INFO] Creating App Service Plan 'ServicePlanb6ba8178-5bbb-49e7'...
[INFO] Successfully created App Service Plan.
[INFO] Creating web App spring-todo-app...
[INFO] Successfully created Web App spring-todo-app.
[INFO] Trying to deploy artifact to spring-todo-app...
[INFO] Successfully deployed the artifact to https://spring-todo-app.azurewebsites.net
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:19 min
[INFO] Finished at: 2019-11-06T15:32:03-07:00
[INFO] Final Memory: 50M/574M
[INFO] -----
```

The output contains the URL to your deployed application (in this example, `https://spring-todo-app.azurewebsites.net`). You can copy this URL into your web browser or run the following command in your Terminal window to load your app.

```
explorer https://spring-todo-app.azurewebsites.net
```

You should see the app running with the remote URL in the address bar:



Stream diagnostic logs

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --application-logging true --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

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

If you don't see console logs immediately, check again in 30 seconds.

NOTE

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

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

Scale out the TODO App

Scale out the application by adding another worker:

```
az appservice plan update --number-of-workers 2 \  
  --name ${WEBAPP_PLAN_NAME} \  
  --resource-group <your-azure-group-name>
```

Clean up resources

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

```
az group delete --name <your-azure-group-name> --yes
```

Next steps

[Azure for Java Developers Spring Boot](#), [Spring Data for Cosmos DB](#), [Azure Cosmos DB](#) and [App Service Linux](#).

Learn more about running Java apps on App Service on Linux in the developer guide.

[Java in App Service Linux dev guide](#)

Tutorial: Create and Manage Linux VMs with the Azure CLI

4/22/2021 • 8 minutes to read • [Edit Online](#)

Azure virtual machines provide a fully configurable and flexible computing environment. This tutorial covers basic Azure virtual machine deployment items such as selecting a VM size, selecting a VM image, and deploying a VM. You learn how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

This tutorial uses the CLI within the [Azure Cloud Shell](#), which is constantly updated to the latest version. To open the Cloud Shell, select **Try it** from the top of any code block.

If you choose to install and use the CLI locally, this tutorial requires that you are running the Azure CLI version 2.0.30 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create resource group

Create a resource group with the `az group create` command.

An Azure resource group is a logical container into which Azure resources are deployed and managed. A resource group must be created before a virtual machine. In this example, a resource group named `myResourceGroupVM` is created in the `eastus` region.

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

The resource group is specified when creating or modifying a VM, which can be seen throughout this tutorial.

Create virtual machine

Create a virtual machine with the `az vm create` command.

When you create a virtual machine, several options are available such as operating system image, disk sizing, and administrative credentials. The following example creates a VM named `myVM` that runs Ubuntu Server. A user account named `azureuser` is created on the VM, and SSH keys are generated if they do not exist in the default key location (`~/ssh`):

```
az vm create \  
  --resource-group myResourceGroupVM \  
  --name myVM \  
  --image UbuntuLTS \  
  --admin-username azureuser \  
  --generate-ssh-keys
```

It may take a few minutes to create the VM. Once the VM has been created, the Azure CLI outputs information about the VM. Take note of the `publicIpAddress`, this address can be used to access the virtual machine..

```
{
  "fqdns": "",
  "id": "/subscriptions/d5b9d4b7-6fc1-0000-0000-000000000000/resourceGroups/myResourceGroupVM/providers/Microsoft.Compute/virtualMachines/myVM",
  "location": "eastus",
  "macAddress": "00-0D-3A-23-9A-49",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "52.174.34.95",
  "resourceGroup": "myResourceGroupVM"
}
```

Connect to VM

You can now connect to the VM with SSH in the Azure Cloud Shell or from your local computer. Replace the example IP address with the `publicIpAddress` noted in the previous step.

```
ssh azureuser@52.174.34.95
```

Once logged in to the VM, you can install and configure applications. When you are finished, you close the SSH session as normal:

```
exit
```

Understand VM images

The Azure marketplace includes many images that can be used to create VMs. In the previous steps, a virtual machine was created using an Ubuntu image. In this step, the Azure CLI is used to search the marketplace for a CentOS image, which is then used to deploy a second virtual machine.

To see a list of the most commonly used images, use the [az vm image list](#) command.

```
az vm image list --output table
```

The command output returns the most popular VM images on Azure.

Offer UrnAlias	Publisher Version	Skus	Urn
WindowsServer Datacenter:latest	MicrosoftWindowsServer Win2016Datacenter	2016-Datacenter latest	MicrosoftWindowsServer:WindowsServer:2016-
WindowsServer Datacenter:latest	MicrosoftWindowsServer Win2012R2Datacenter	2012-R2-Datacenter latest	MicrosoftWindowsServer:WindowsServer:2012-R2-
WindowsServer SP1:latest	MicrosoftWindowsServer Win2008R2SP1	2008-R2-SP1 latest	MicrosoftWindowsServer:WindowsServer:2008-R2-
WindowsServer Datacenter:latest	MicrosoftWindowsServer Win2012Datacenter	2012-Datacenter latest	MicrosoftWindowsServer:WindowsServer:2012-
UbuntuServer UbuntuLTS	Canonical latest	16.04-LTS	Canonical:UbuntuServer:16.04-LTS:latest
CentOS	OpenLogic	7.3	OpenLogic:CentOS:7.3:latest
CentOS	latest		
openSUSE-Leap	SUSE	42.2	SUSE:openSUSE-Leap:42.2:latest
openSUSE-Leap	latest		
RHEL	RedHat	7.3	RedHat:RHEL:7.3:latest
RHEL	latest		
SLES	SUSE	12-SP2	SUSE:SLES:12-SP2:latest
SLES	latest		
Debian	credativ	8	credativ:Debian:8:latest
Debian	latest		
CoreOS	CoreOS	Stable	CoreOS:CoreOS:Stable:latest
CoreOS	latest		

A full list can be seen by adding the `--all` argument. The image list can also be filtered by `--publisher` or `--offer`. In this example, the list is filtered for all images with an offer that matches *CentOS*.

```
az vm image list --offer CentOS --all --output table
```

Partial output:

Offer	Publisher	Skus	Urn	Version
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.201501	6.5.201501
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.201503	6.5.201503
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.201506	6.5.201506
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.20150904	6.5.20150904
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.20160309	6.5.20160309
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.20170207	6.5.20170207

To deploy a VM using a specific image, take note of the value in the *Urn* column, which consists of the publisher, offer, SKU, and optionally a version number to [identify](#) the image. When specifying the image, the image version number can be replaced with "latest", which selects the latest version of the distribution. In this example, the `--image` argument is used to specify the latest version of a CentOS 6.5 image.

```
az vm create --resource-group myResourceGroupVM --name myVM2 --image OpenLogic:CentOS:6.5:latest --generate-ssh-keys
```

Understand VM sizes

A virtual machine size determines the amount of compute resources such as CPU, GPU, and memory that are made available to the virtual machine. Virtual machines need to be sized appropriately for the expected workload. If workload increases, an existing virtual machine can be resized.

VM Sizes

The following table categorizes sizes into use cases.

TYPE	COMMON SIZES	DESCRIPTION
General purpose	B, Dsv3, Dv3, DSv2, Dv2, Av2, DC	Balanced CPU-to-memory. Ideal for dev / test and small to medium applications and data solutions.
Compute optimized	Fsv2	High CPU-to-memory. Good for medium traffic applications, network appliances, and batch processes.
Memory optimized	Esv3, Ev3, M, DSv2, Dv2	High memory-to-core. Great for relational databases, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2, Ls	High disk throughput and IO. Ideal for Big Data, SQL, and NoSQL databases.
GPU	NV, NVv2, NC, NCv2, NCv3, ND	Specialized VMs targeted for heavy graphic rendering and video editing.
High performance	H	Our most powerful CPU VMs with optional high-throughput network interfaces (RDMA).

Find available VM sizes

To see a list of VM sizes available in a particular region, use the [az vm list-sizes](#) command.

```
az vm list-sizes --location eastus --output table
```

Partial output:

MaxDataDiskCount	MemoryInMb	Name	NumberOfCores	OsDiskSizeInMb
ResourceDiskSizeInMb				
-----	-----	-----	-----	-----

7168	2	3584 Standard_DS1	1	1047552
14336	4	7168 Standard_DS2	2	1047552
28672	8	14336 Standard_DS3	4	1047552
57344	16	28672 Standard_DS4	8	1047552
28672	4	14336 Standard_DS11	2	1047552
57344	8	28672 Standard_DS12	4	1047552
114688	16	57344 Standard_DS13	8	1047552
229376	32	114688 Standard_DS14	16	1047552
20480	1	768 Standard_A0	1	1047552
71680	2	1792 Standard_A1	1	1047552
138240	4	3584 Standard_A2	2	1047552
291840	8	7168 Standard_A3	4	1047552
138240	4	14336 Standard_A5	2	1047552
619520	16	14336 Standard_A4	8	1047552
291840	8	28672 Standard_A6	4	1047552
619520	16	57344 Standard_A7	8	1047552

Create VM with specific size

In the previous VM creation example, a size was not provided, which results in a default size. A VM size can be selected at creation time using `az vm create` and the `--size` argument.

```
az vm create \
  --resource-group myResourceGroupVM \
  --name myVM3 \
  --image UbuntuLTS \
  --size Standard_F4s \
  --generate-ssh-keys
```

Resize a VM

After a VM has been deployed, it can be resized to increase or decrease resource allocation. You can view the current of size of a VM with `az vm show`:

```
az vm show --resource-group myResourceGroupVM --name myVM --query hardwareProfile.vmSize
```

Before resizing a VM, check if the desired size is available on the current Azure cluster. The `az vm list-vm-resize-options` command returns the list of sizes.

```
az vm list-vm-resize-options --resource-group myResourceGroupVM --name myVM --query [].name
```

If the desired size is available, the VM can be resized from a powered-on state, however it is rebooted during the operation. Use the [az vm resize](#) command to perform the resize.

```
az vm resize --resource-group myResourceGroupVM --name myVM --size Standard_DS4_v2
```

If the desired size is not on the current cluster, the VM needs to be deallocated before the resize operation can occur. Use the [az vm deallocate](#) command to stop and deallocate the VM. Note, when the VM is powered back on, any data on the temp disk may be removed. The public IP address also changes unless a static IP address is being used.

```
az vm deallocate --resource-group myResourceGroupVM --name myVM
```

Once deallocated, the resize can occur.

```
az vm resize --resource-group myResourceGroupVM --name myVM --size Standard_GS1
```

After the resize, the VM can be started.

```
az vm start --resource-group myResourceGroupVM --name myVM
```

VM power states

An Azure VM can have one of many power states. This state represents the current state of the VM from the standpoint of the hypervisor.

Power states

POWER STATE	DESCRIPTION
Starting	Indicates the virtual machine is being started.
Running	Indicates that the virtual machine is running.
Stopping	Indicates that the virtual machine is being stopped.
Stopped	Indicates that the virtual machine is stopped. Virtual machines in the stopped state still incur compute charges.
Deallocating	Indicates that the virtual machine is being deallocated.
Deallocated	Indicates that the virtual machine is removed from the hypervisor but still available in the control plane. Virtual machines in the Deallocated state do not incur compute charges.
-	Indicates that the power state of the virtual machine is unknown.

Find the power state

To retrieve the state of a particular VM, use the [az vm get-instance-view](#) command. Be sure to specify a valid name for a virtual machine and resource group.


```
az vm get-instance-view \  
  --name myVM \  
  --resource-group myResourceGroupVM \  
  --query instanceView.statuses[1] --output table
```

Output:

```
ode           DisplayStatus   Level  
-----  
PowerState/running VM running   Info
```

To retrieve the power state of all the VMs in your subscription, use the [Virtual Machines - List All API](#) with parameter `statusOnly` set to `true`.

Management tasks

During the life-cycle of a virtual machine, you may want to run management tasks such as starting, stopping, or deleting a virtual machine. Additionally, you may want to create scripts to automate repetitive or complex tasks. Using the Azure CLI, many common management tasks can be run from the command line or in scripts.

Get IP address

This command returns the private and public IP addresses of a virtual machine.

```
az vm list-ip-addresses --resource-group myResourceGroupVM --name myVM --output table
```

Stop virtual machine

```
az vm stop --resource-group myResourceGroupVM --name myVM
```

Start virtual machine

```
az vm start --resource-group myResourceGroupVM --name myVM
```

Delete resource group

Deleting a resource group also deletes all resources contained within, such as the VM, virtual network, and disk.

The `--no-wait` parameter returns control to the prompt without waiting for the operation to complete. The

`--yes` parameter confirms that you wish to delete the resources without an additional prompt to do so.

```
az group delete --name myResourceGroupVM --no-wait --yes
```

Next steps

In this tutorial, you learned about basic VM creation and management such as how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

Advance to the next tutorial to learn about VM disks.

Tutorial: Create and Manage Windows VMs with Azure PowerShell

3/10/2021 • 7 minutes to read • [Edit Online](#)

Azure virtual machines provide a fully configurable and flexible computing environment. This tutorial covers basic Azure virtual machine (VM) deployment tasks like selecting a VM size, selecting a VM image, and deploying a VM. You learn how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

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.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/powershell>. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

Create resource group

Create a resource group with the [New-AzResourceGroup](#) command.

An Azure resource group is a logical container into which Azure resources are deployed and managed. A resource group must be created before a virtual machine. In the following example, a resource group named *myResourceGroupVM* is created in the *EastUS* region:

```
New-AzResourceGroup `
  -ResourceGroupName "myResourceGroupVM" `
  -Location "EastUS"
```

The resource group is specified when creating or modifying a VM, which can be seen throughout this tutorial.

Create a VM

When creating a VM, several options are available like operating system image, network configuration, and administrative credentials. This example creates a VM named *myVM*, running the default version of Windows Server 2016 Datacenter.

Set the username and password needed for the administrator account on the VM with [Get-Credential](#):

```
$cred = Get-Credential
```

Create the VM with [New-AzVM](#).

```
New-AzVm `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" `
  -Location "EastUS" `
  -VirtualNetworkName "myVnet" `
  -SubnetName "mySubnet" `
  -SecurityGroupName "myNetworkSecurityGroup" `
  -PublicIpAddressName "myPublicIpAddress" `
  -Credential $cred
```

Connect to VM

After the deployment has completed, create a remote desktop connection with the VM.

Run the following commands to return the public IP address of the VM. Take note of this IP Address so you can connect to it with your browser to test web connectivity in a future step.

```
Get-AzPublicIpAddress `
  -ResourceGroupName "myResourceGroupVM" | Select IPAddress
```

Use the following command, on your local machine, to create a remote desktop session with the VM. Replace the IP address with the *publicIpAddress* of your VM. When prompted, enter the credentials used when creating the VM.

```
mstsc /v:<publicIpAddress>
```

In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username and password you created for the VM and then click **OK**.

Understand marketplace images

The Azure marketplace includes many images that can be used to create a new VM. In the previous steps, a VM was created using the Windows Server 2016 Datacenter image. In this step, the PowerShell module is used to search the marketplace for other Windows images, which can also be used as a base for new VMs. This process consists of finding the publisher, offer, SKU, and optionally a version number to **identify** the image.

Use the [Get-AzVMImagePublisher](#) command to return a list of image publishers:

```
Get-AzVMImagePublisher -Location "EastUS"
```

Use the [Get-AzVMImageOffer](#) to return a list of image offers. With this command, the returned list is filtered on the specified publisher named `MicrosoftWindowsServer`:

```
Get-AzVMImageOffer `
  -Location "EastUS" `
  -PublisherName "MicrosoftWindowsServer"
```

The results will look something like this example:

Offer	PublisherName	Location
Windows-HUB	MicrosoftWindowsServer	EastUS
WindowsServer	MicrosoftWindowsServer	EastUS
WindowsServer-HUB	MicrosoftWindowsServer	EastUS

The `Get-AzVMImageSku` command will then filter on the publisher and offer name to return a list of image names.

```
Get-AzVMImageSku `
  -Location "EastUS" `
  -PublisherName "MicrosoftWindowsServer" `
  -Offer "WindowsServer"
```

The results will look something like this example:

Skus	Offer	PublisherName	Location
2008-R2-SP1	WindowsServer	MicrosoftWindowsServer	EastUS
2008-R2-SP1-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2012-Datacenter	WindowsServer	MicrosoftWindowsServer	EastUS
2012-Datacenter-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2012-R2-Datacenter	WindowsServer	MicrosoftWindowsServer	EastUS
2012-R2-Datacenter-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-Server-Core	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-Server-Core-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-with-Containers	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-with-Containers-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-with-RDSH	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Nano-Server	WindowsServer	MicrosoftWindowsServer	EastUS

This information can be used to deploy a VM with a specific image. This example deploys a VM using the latest version of a Windows Server 2016 with Containers image.

```
New-AzVm `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM2" `
  -Location "EastUS" `
  -VirtualNetworkName "myVnet" `
  -SubnetName "mySubnet" `
  -SecurityGroupName "myNetworkSecurityGroup" `
  -PublicIpAddressName "myPublicIpAddress2" `
  -ImageName "MicrosoftWindowsServer:WindowsServer:2016-Datacenter-with-Containers:latest" `
  -Credential $cred `
  -AsJob
```

The `-AsJob` parameter creates the VM as a background task, so the PowerShell prompts return to you. You can view details of background jobs with the `Get-Job` cmdlet.

Understand VM sizes

The VM size determines the amount of compute resources like CPU, GPU, and memory that are made available to the VM. Virtual machines should be created using a VM size appropriate for the workload. If a workload increases, an existing virtual machine can also be resized.

VM Sizes

The following table categorizes sizes into use cases.

TYPE	COMMON SIZES	DESCRIPTION
General purpose	B, Dsv3, Dv3, DSv2, Dv2, Av2, DC	Balanced CPU-to-memory. Ideal for dev / test and small to medium applications and data solutions.
Compute optimized	Fsv2	High CPU-to-memory. Good for medium traffic applications, network appliances, and batch processes.
Memory optimized	Esv3, Ev3, M, DSv2, Dv2	High memory-to-core. Great for relational databases, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2, Ls	High disk throughput and IO. Ideal for Big Data, SQL, and NoSQL databases.
GPU	NV, NVv2, NC, NCv2, NCv3, ND	Specialized VMs targeted for heavy graphic rendering and video editing.
High performance	H	Our most powerful CPU VMs with optional high-throughput network interfaces (RDMA).

Find available VM sizes

To see a list of VM sizes available in a particular region, use the [Get-AzVMSize](#) command.

```
Get-AzVMSize -Location "EastUS"
```

Resize a VM

After a VM has been deployed, it can be resized to increase or decrease resource allocation.

Before resizing a VM, check if the size you want is available on the current VM cluster. The [Get-AzVMSize](#) command returns a list of sizes.

```
Get-AzVMSize -ResourceGroupName "myResourceGroupVM" -VMName "myVM"
```

If the size is available, the VM can be resized from a powered-on state, however it is rebooted during the operation.

```
$vm = Get-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -VMName "myVM"
$vm.HardwareProfile.VmSize = "Standard_DS3_v2"
Update-AzVM `
  -VM $vm `
  -ResourceGroupName "myResourceGroupVM"
```

If the size you want isn't available on the current cluster, the VM needs to be deallocated before the resize operation can occur. Deallocating a VM will remove any data on the temp disk, and the public IP address will change unless a static IP address is being used.

```

Stop-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" -Force
$vm = Get-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -VMName "myVM"
$vm.HardwareProfile.VmSize = "Standard_E2s_v3"
Update-AzVM -VM $vm `
  -ResourceGroupName "myResourceGroupVM"
Start-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name $vm.name

```

VM power states

An Azure VM can have one of many power states.

POWER STATE	DESCRIPTION
Starting	The virtual machine is being started.
Running	The virtual machine is running.
Stopping	The virtual machine is being stopped.
Stopped	The VM is stopped. Virtual machines in the stopped state still incur compute charges.
Deallocating	The VM is being deallocated.
Deallocated	Indicates that the VM is removed from the hypervisor but is still available in the control plane. Virtual machines in the <code>Deallocated</code> state do not incur compute charges.
-	The power state of the VM is unknown.

To get the state of a particular VM, use the [Get-AzVM](#) command. Be sure to specify a valid name for a VM and resource group.

```

Get-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" `
  -Status | Select @{n="Status"; e={$_.Statuses[1].Code}}

```

The output will look something like this example:

```

Status
-----
PowerState/running

```

To retrieve the power state of all the VMs in your subscription, use the [Virtual Machines - List All API](#) with parameter `statusOnly` set to `true`.

Management tasks

During the lifecycle of a VM, you may want to run management tasks like starting, stopping, or deleting a VM. Additionally, you may want to create scripts to automate repetitive or complex tasks. Using Azure PowerShell, many common management tasks can be run from the command line or in scripts.

Stop a VM

Stop and deallocate a VM with [Stop-AzVM](#):

```
Stop-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" -Force
```

If you want to keep the VM in a provisioned state, use the `-StayProvisioned` parameter.

Start a VM

```
Start-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM"
```

Delete resource group

Everything inside of a resource group is deleted when you delete the resource group.

```
Remove-AzResourceGroup `
  -Name "myResourceGroupVM" `
  -Force
```

Next steps

In this tutorial, you learned about basic VM creation and management such as how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

Advance to the next tutorial to learn about VM disks.

[Create and Manage VM disks](#)

Create a function triggered by Azure Queue storage

11/2/2020 • 5 minutes to read • [Edit Online](#)

Learn how to create a function that is triggered when messages are submitted to an Azure Storage queue.

Prerequisites

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

Create an Azure Function app

1. From the Azure portal menu or the **Home** page, select **Create a resource**.
2. In the **New** page, select **Compute** > **Function App**.
3. On the **Basics** page, use the function app settings as specified in the following table.

SETTING	SUGGESTED VALUE	DESCRIPTION
Subscription	Your subscription	The subscription under which this new function app is created.
Resource Group	<i>myResourceGroup</i>	Name for the new resource group in which to create your function app.
Function App name	Globally unique name	Name that identifies your new function app. Valid characters are <code>a-z</code> (case insensitive), <code>0-9</code> , and <code>-</code> .
Publish	Code	Option to publish code files or a Docker container.
Runtime stack	Preferred language	Choose a runtime that supports your favorite function programming language. In-portal editing is only available for JavaScript, PowerShell, TypeScript, and C# script. C# class library, Java, and Python functions must be developed locally .
Version	Version number	Choose the version of your installed runtime.
Region	Preferred region	Choose a region near you or near other services your functions access.

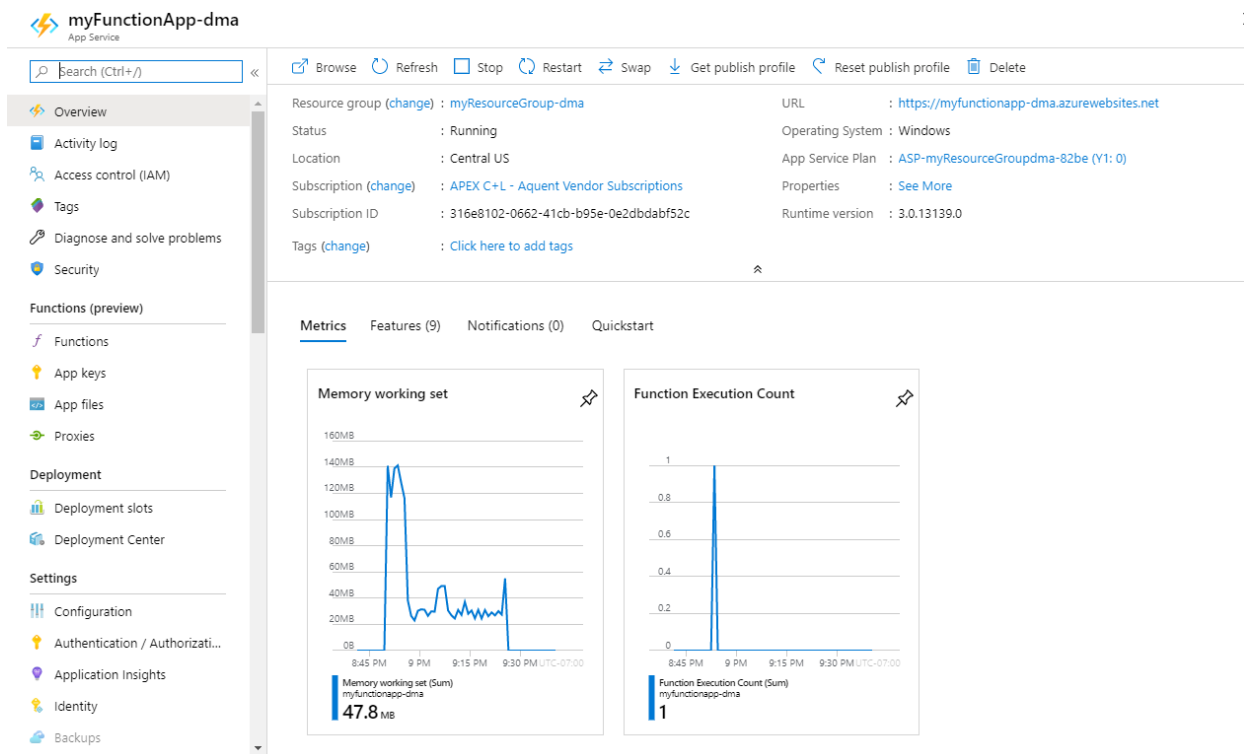
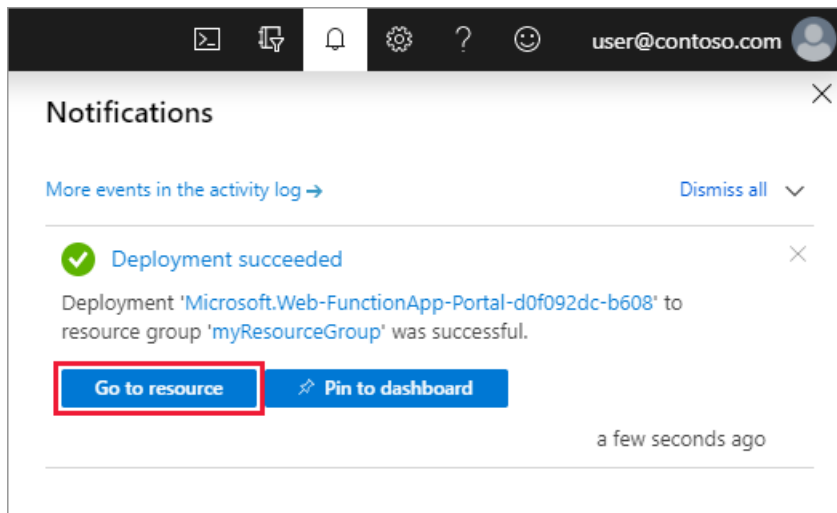
4. Select **Next : Hosting**. On the **Hosting** page, enter the following settings.

SETTING	SUGGESTED VALUE	DESCRIPTION
Storage account	Globally unique name	Create a storage account used by your function app. Storage account names must be between 3 and 24 characters in length and can contain numbers and lowercase letters only. You can also use an existing account, which must meet the storage account requirements .
Operating system	Windows	An operating system is pre-selected for you based on your runtime stack selection, but you can change the setting if necessary. In-portal editing is only supported on Windows.
Plan	Consumption (Serverless)	Hosting plan that defines how resources are allocated to your function app. In the default Consumption plan, resources are added dynamically as required by your functions. In this serverless hosting, you pay only for the time your functions run. When you run in an App Service plan, you must manage the scaling of your function app .

5. Select **Next : Monitoring**. On the **Monitoring** page, enter the following settings.

SETTING	SUGGESTED VALUE	DESCRIPTION
Application Insights	Default	Creates an Application Insights resource of the same <i>App name</i> in the nearest supported region. By expanding this setting or selecting Create new , you can change the Application Insights name or choose a different region in an Azure geography where you want to store your data.

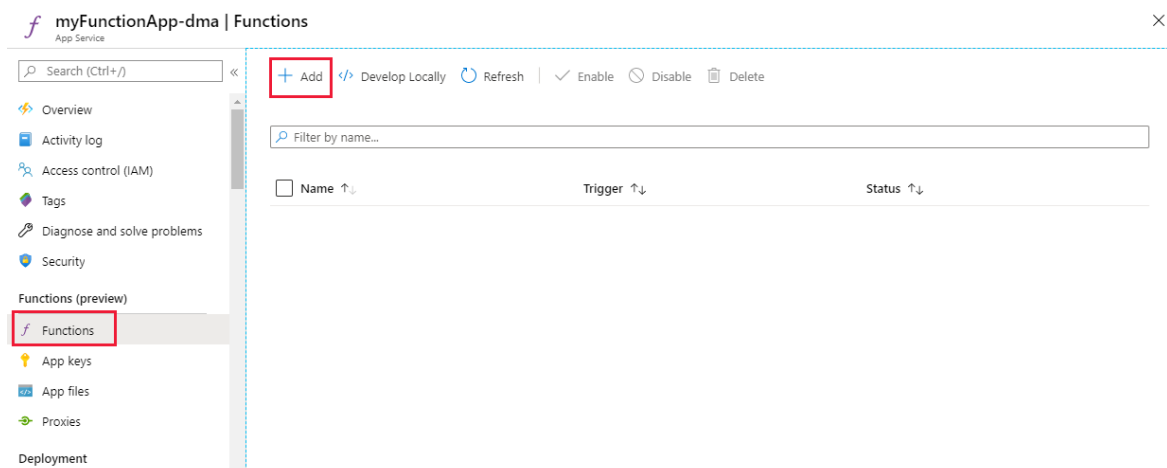
6. Select **Review + create** to review the app configuration selections.
7. On the **Review + create** page, review your settings, and then select **Create** to provision and deploy the function app.
8. Select the **Notifications** icon in the upper-right corner of the portal and watch for the **Deployment succeeded** message.
9. Select **Go to resource** to view your new function app. You can also select **Pin to dashboard**. Pinning makes it easier to return to this function app resource from your dashboard.



Next, you create a function in the new function app.

Create a Queue triggered function

1. Select Functions, and then select + Add to add a new function.



2. Choose the **Azure Queue Storage trigger** template.
3. Use the settings as specified in the table below the image.

New Function

Create a new function in this Function App. Start by selecting a template below.

[Templates](#)

[Details](#)

New Function *

Queue name * ⓘ

Storage account connection * ⓘ

[New](#)

[Create Function](#)

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	Unique in your function app	Name of this queue triggered function.
Queue name	myqueue-items	Name of the queue to connect to in your Storage account.
Storage account connection	AzureWebJobsStorage	You can use the storage account connection already being used by your function app, or create a new one.

4. Select **Create Function** to create your function.

New Function

Create a new function in this Function App. Start by selecting a template below.

[Templates](#)

[Details](#)

New Function *

Queue name * ⓘ

Storage account connection * ⓘ

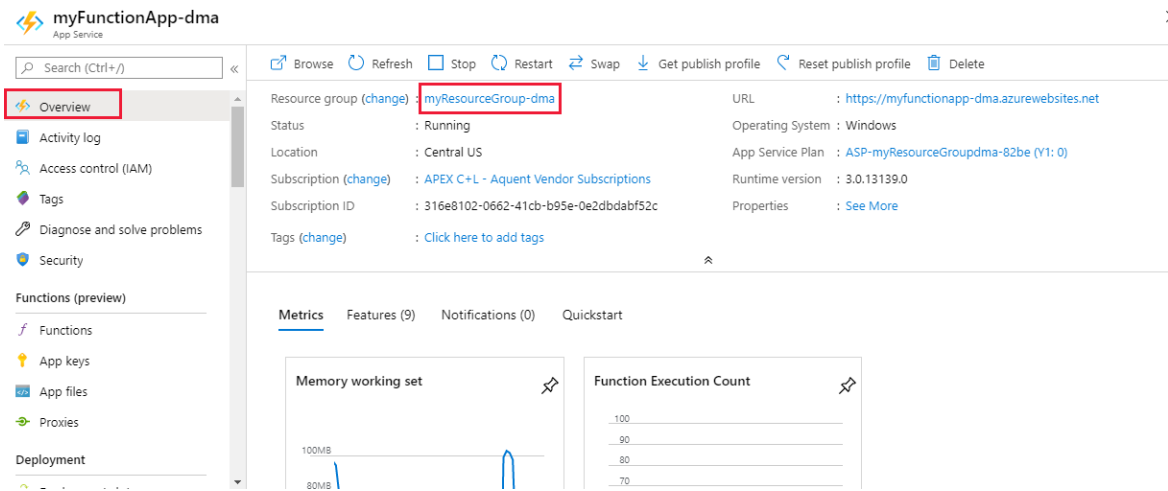
[New](#)

[Create Function](#)

Next, you connect to your Azure storage account and create the `myqueue-items` storage queue.

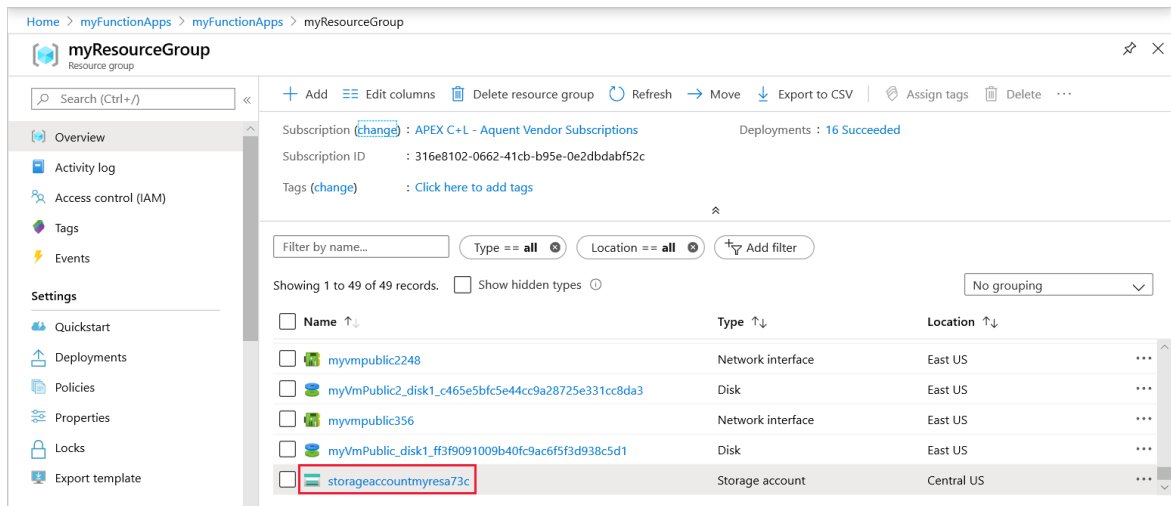
Create the queue

1. In your function, on the **Overview** page, select your resource group.



The screenshot shows the Azure App Service Overview page for 'myFunctionApp-dma'. The left sidebar has 'Overview' selected. The main area displays the resource group 'myResourceGroup-dma' (highlighted with a red box). Below this, there are two charts: 'Memory working set' and 'Function Execution Count'.

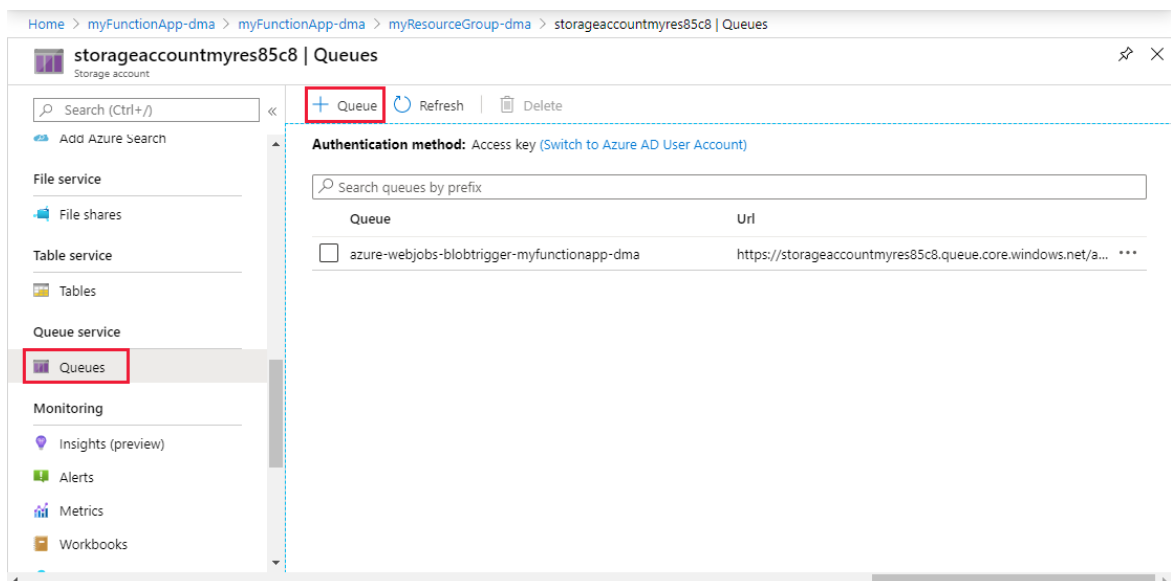
2. Find and select your resource group's storage account.



The screenshot shows the Azure Resource Group 'myResourceGroup' page. The left sidebar has 'Overview' selected. A table lists resources, with 'storageaccountmyres73c' highlighted in red.

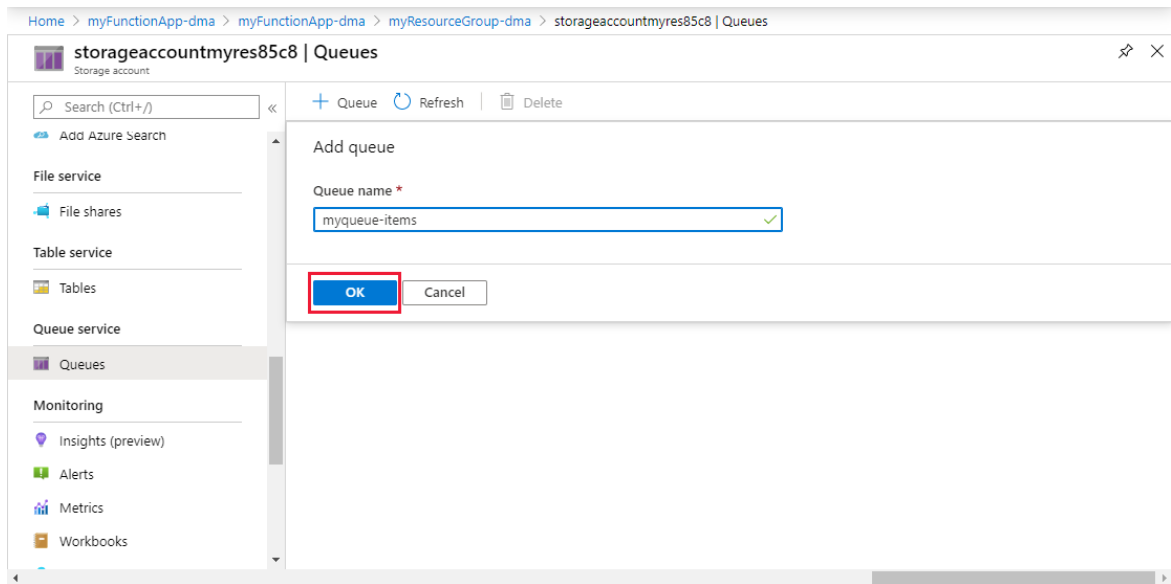
Name	Type	Location
myvmpublic2248	Network interface	East US
myVmPublic2_disk1_c465e5bfc5e44cc9a28725e331cc8da3	Disk	East US
myvmpublic356	Network interface	East US
myVmPublic_disk1_ff3f9091009b40fc9ac6f5f3d938c5d1	Disk	East US
storageaccountmyres73c	Storage account	Central US

3. Choose **Queues**, and then choose **+ Queue**.



The screenshot shows the Azure Storage account 'storageaccountmyres85c8' page. The left sidebar has 'Queues' selected. The '+ Queue' button is highlighted in red.

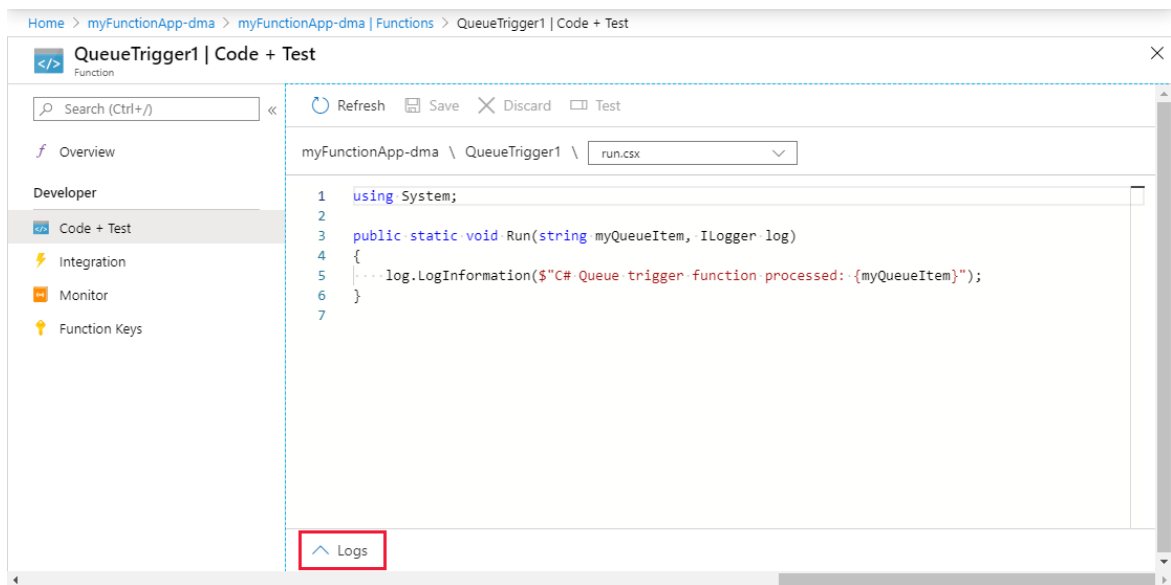
4. In the **Name** field, type `myqueue-items`, and then select **Create**.



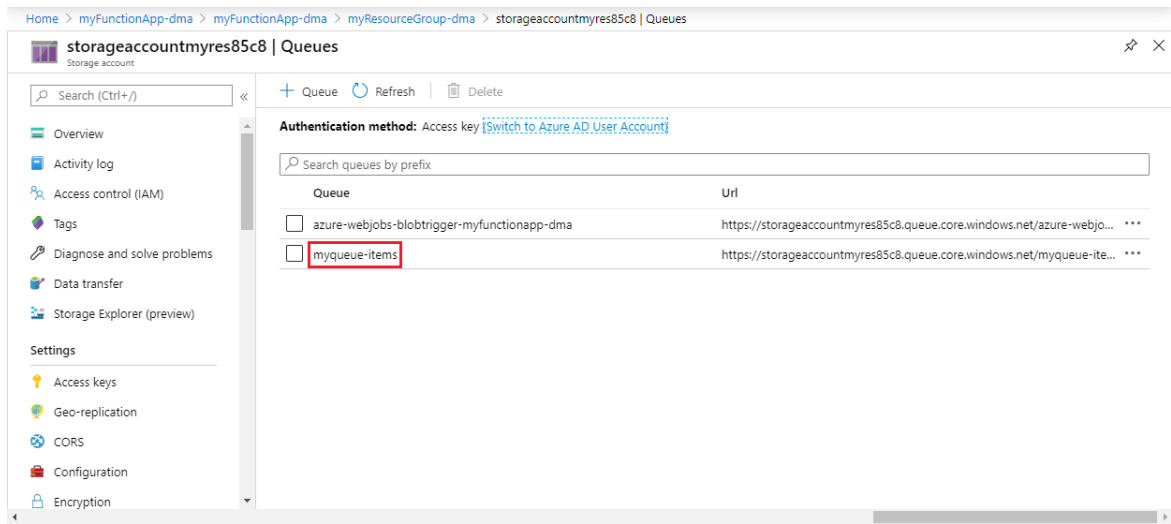
Now that you have a storage queue, you can test the function by adding a message to the queue.

Test the function

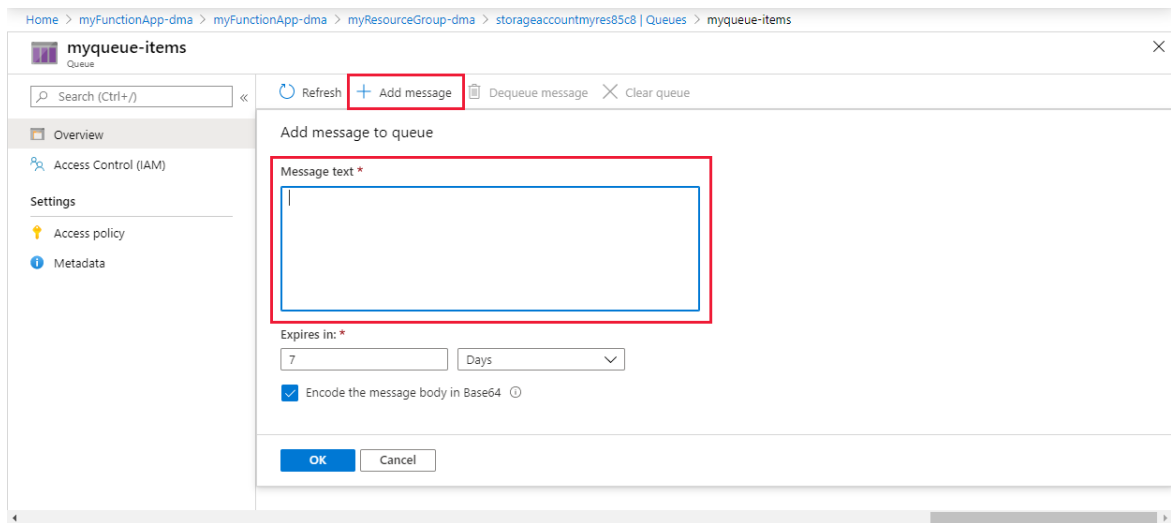
1. Back in the Azure portal, browse to your function expand the **Logs** at the bottom of the page and make sure that log streaming isn't paused.



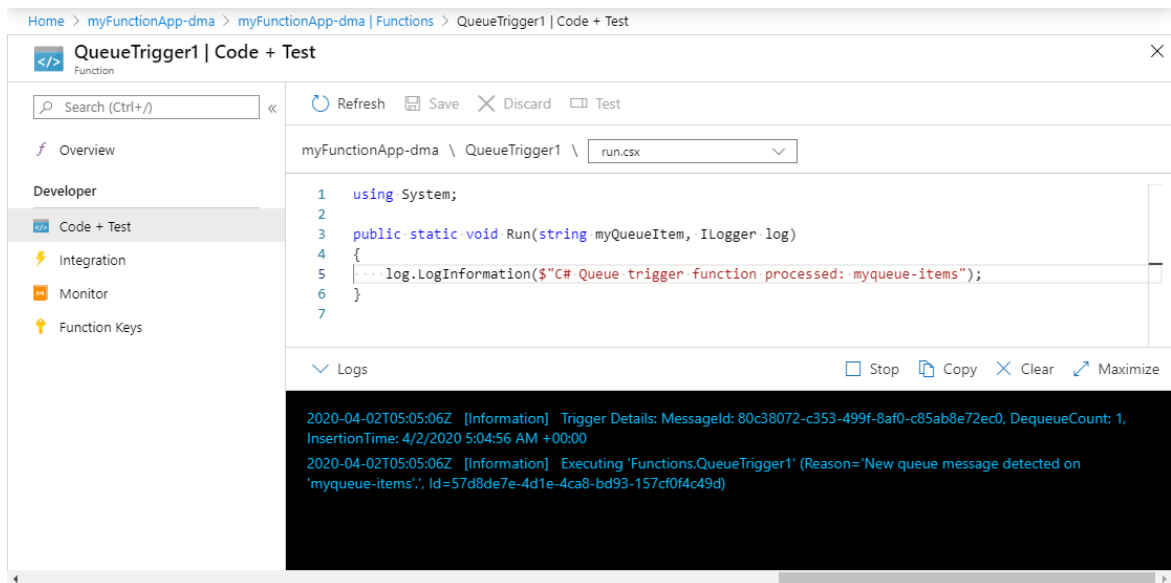
2. In a separate browser window, go to your resource group in the Azure portal, and select the storage account.
3. Select **Queues**, and then select the **myqueue-items** container.



4. Select **Add message**, and type "Hello World!" in **Message text**. Select **OK**.



5. Wait for a few seconds, then go back to your function logs and verify that the new message has been read from the queue.



6. Back in your storage queue, select **Refresh** and verify that the message has been processed and is no longer in the queue.

Clean up resources

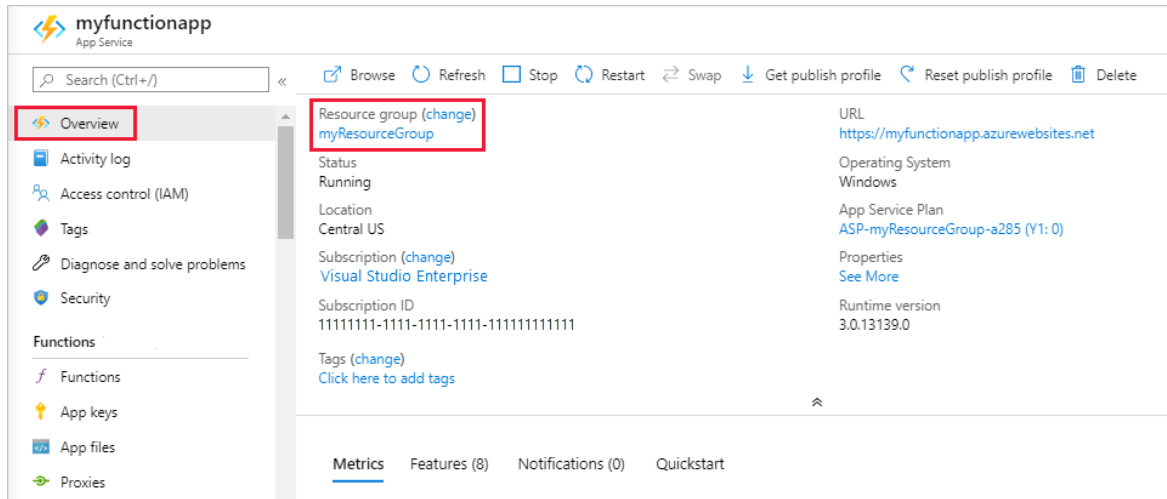
Other quickstarts in this collection build upon this quickstart. If you plan to work with subsequent quickstarts, tutorials, or with any of the services you have created in this quickstart, do not clean up the resources.

Resources in Azure refer to function apps, functions, storage accounts, and so forth. They're grouped into *resource groups*, and you can delete everything in a group by deleting the group.

You created resources to complete these quickstarts. You may be billed for these resources, depending on your [account status](#) and [service pricing](#). If you don't need the resources anymore, here's how to delete them:

1. In the Azure portal, go to the **Resource group** page.

To get to that page from the function app page, select the **Overview** tab and then select the link under **Resource group**.



To get to that page from the dashboard, select **Resource groups**, and then select the resource group that you used for this article.

2. In the **Resource group** page, review the list of included resources, and verify that they're the ones you want to delete.
3. Select **Delete resource group**, and follow the instructions.

Deletion may take a couple of minutes. When it's done, a notification appears for a few seconds. You can also select the bell icon at the top of the page to view the notification.

Next steps

You have created a function that runs when a message is added to a storage queue. For more information about Queue storage triggers, see [Azure Functions Storage queue bindings](#).

Now that you have a created your first function, let's add an output binding to the function that writes a message back to another queue.

[Add messages to an Azure Storage queue using Functions](#)

Run a custom container in Azure

3/5/2021 • 7 minutes to read • [Edit Online](#)

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

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

NOTE

Windows Containers is limited to Azure Files and does not currently support Azure Blob.

Prerequisites

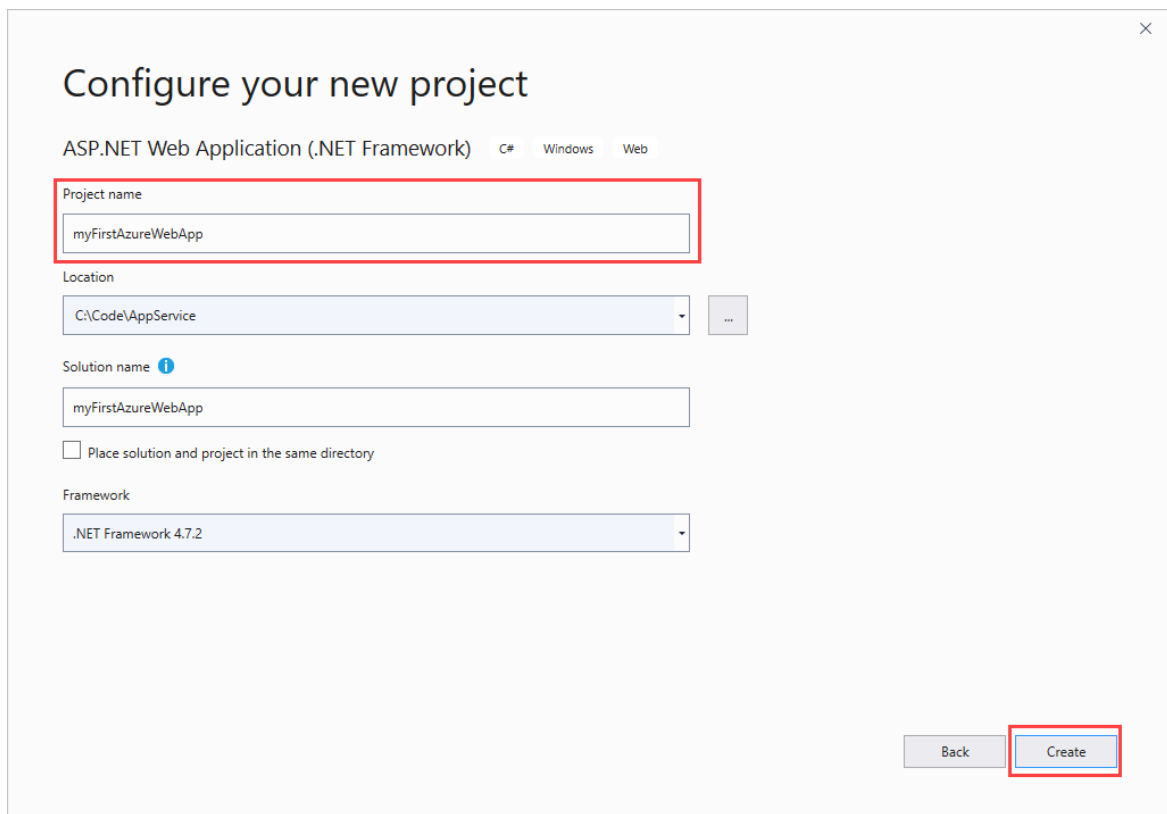
To complete this tutorial:

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

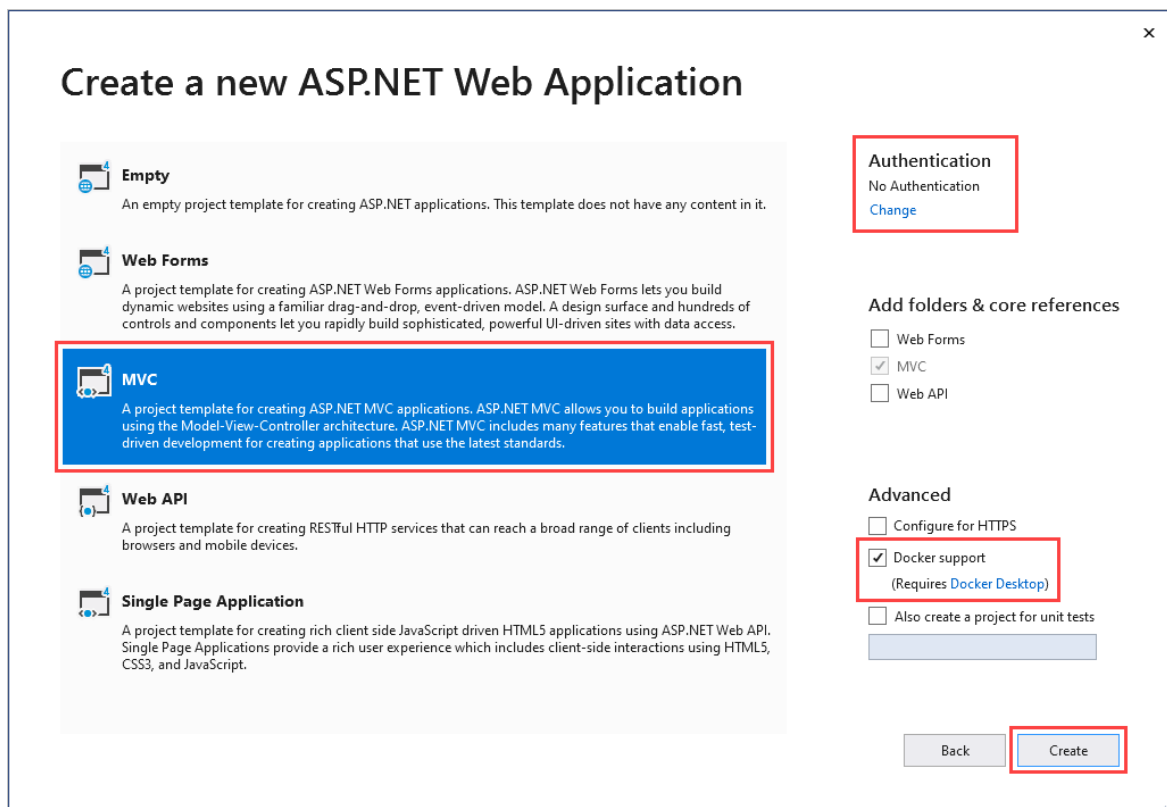
Create an ASP.NET web app

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

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



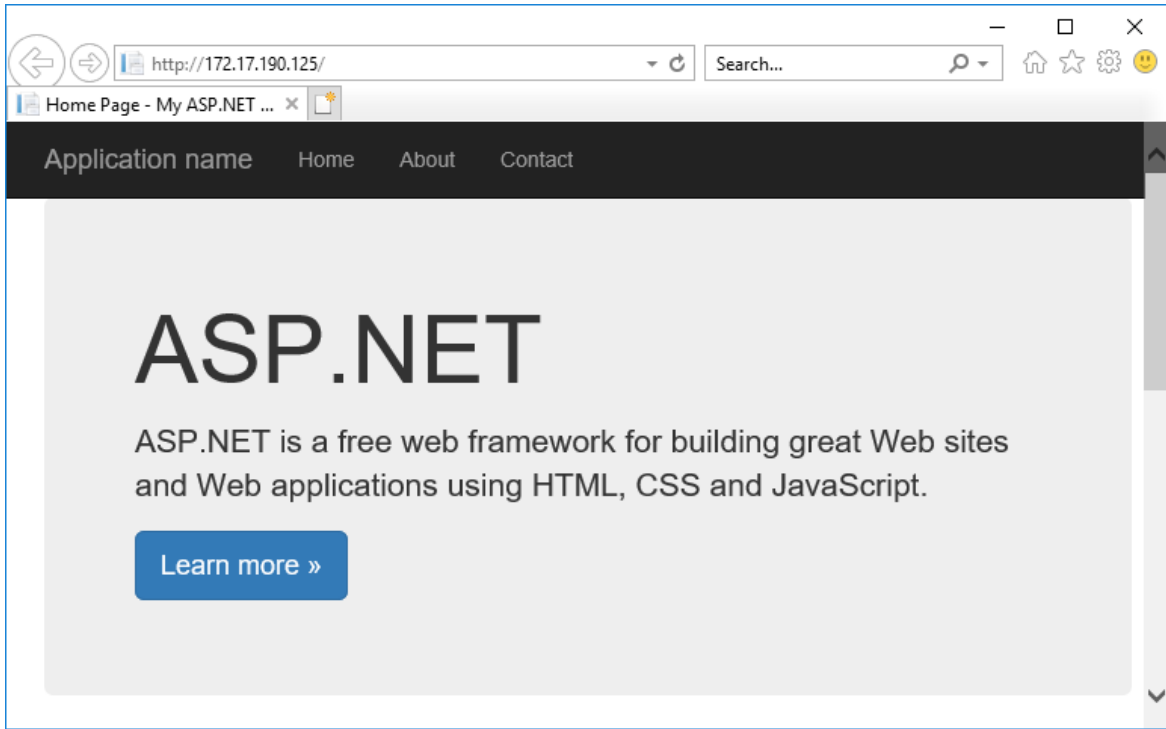
4. You can deploy any type of ASP.NET web app to Azure. For this quickstart, choose the **MVC** template.
5. Select **Docker support**, and make sure authentication is set to **No Authentication**. Select **Create**.



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

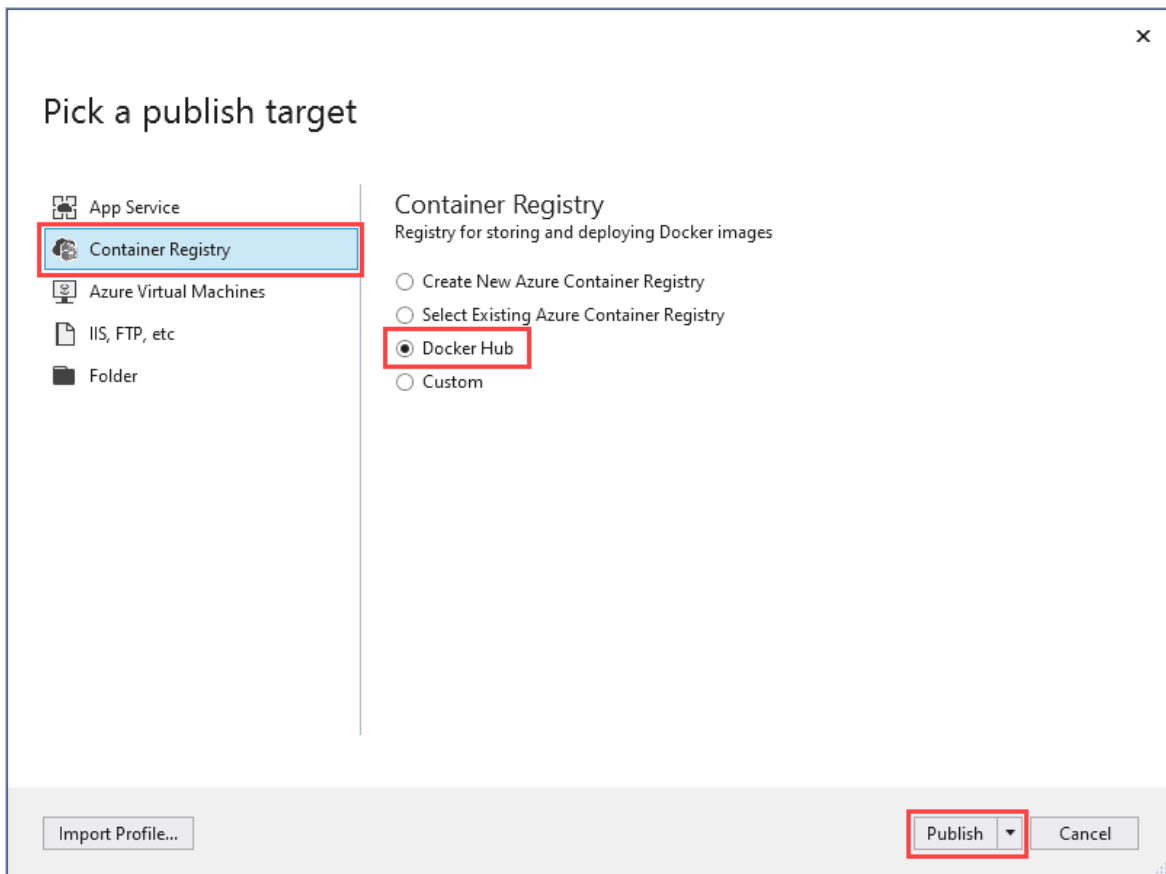
```
FROM mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-1tsc2019
```

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



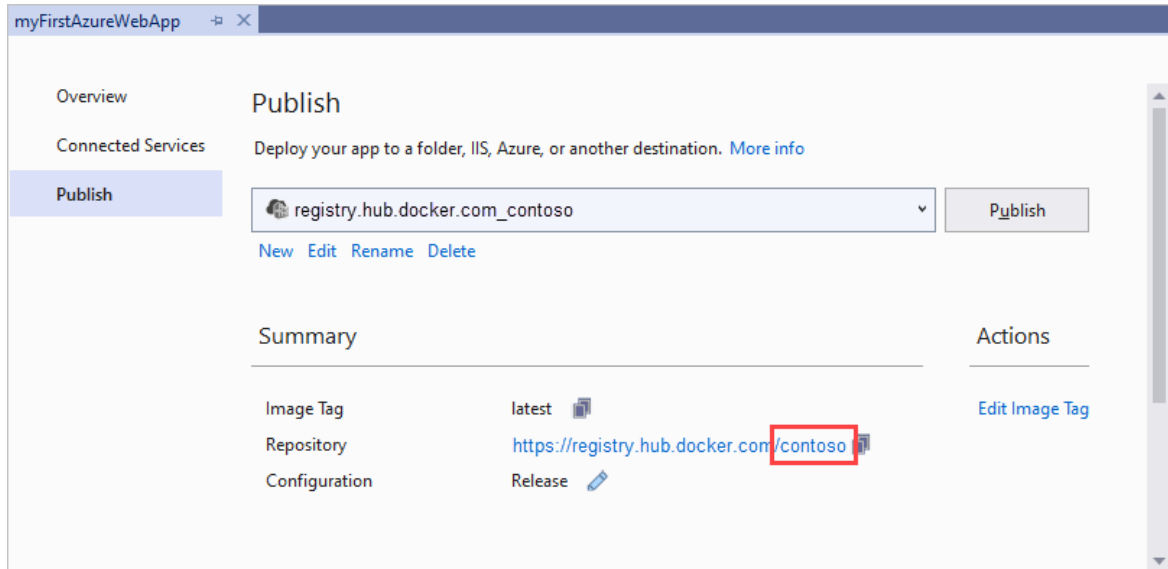
Publish to Docker Hub

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



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

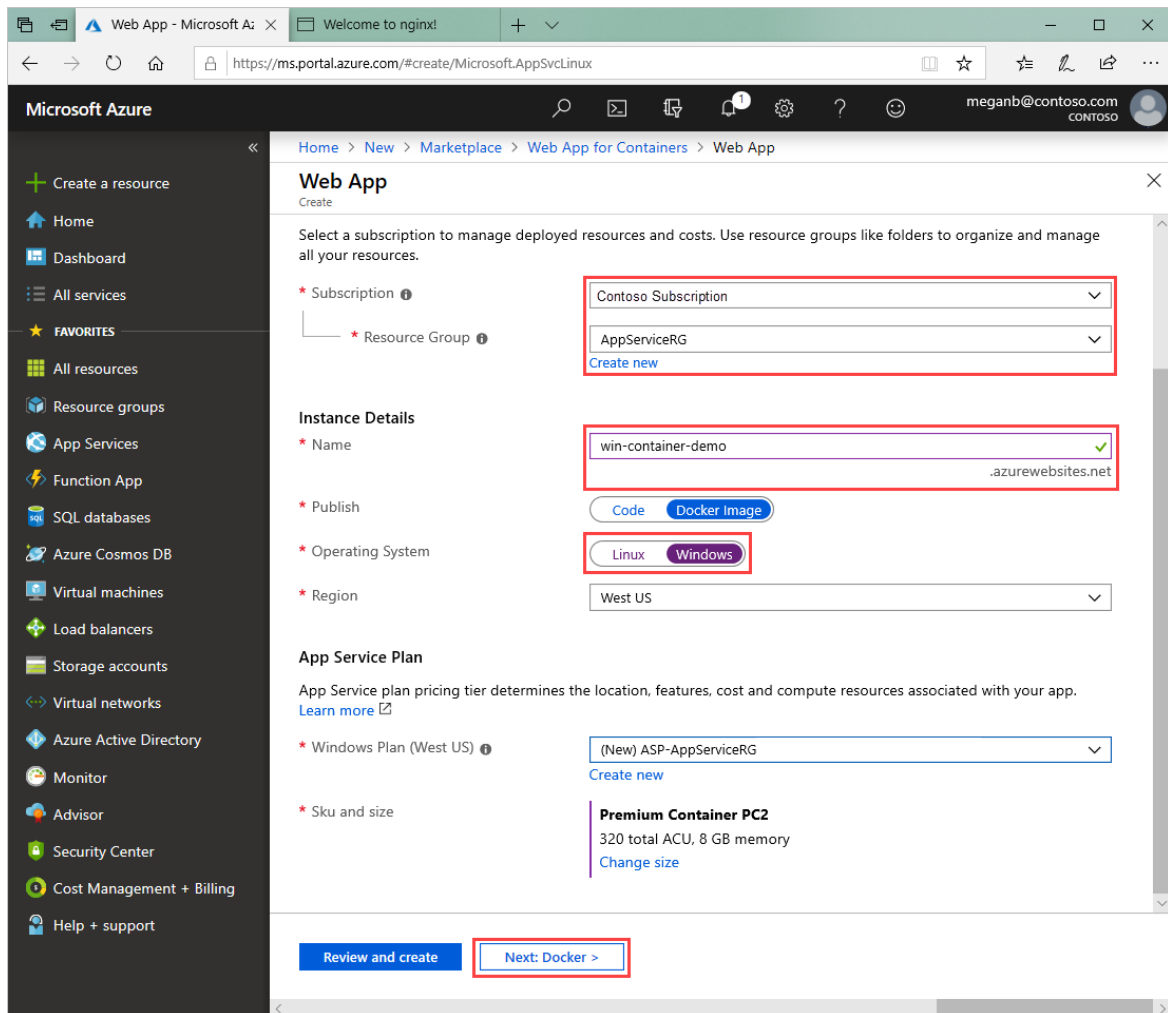
Wait for the deployment to complete. The **Publish** page now shows the repository name to use later.



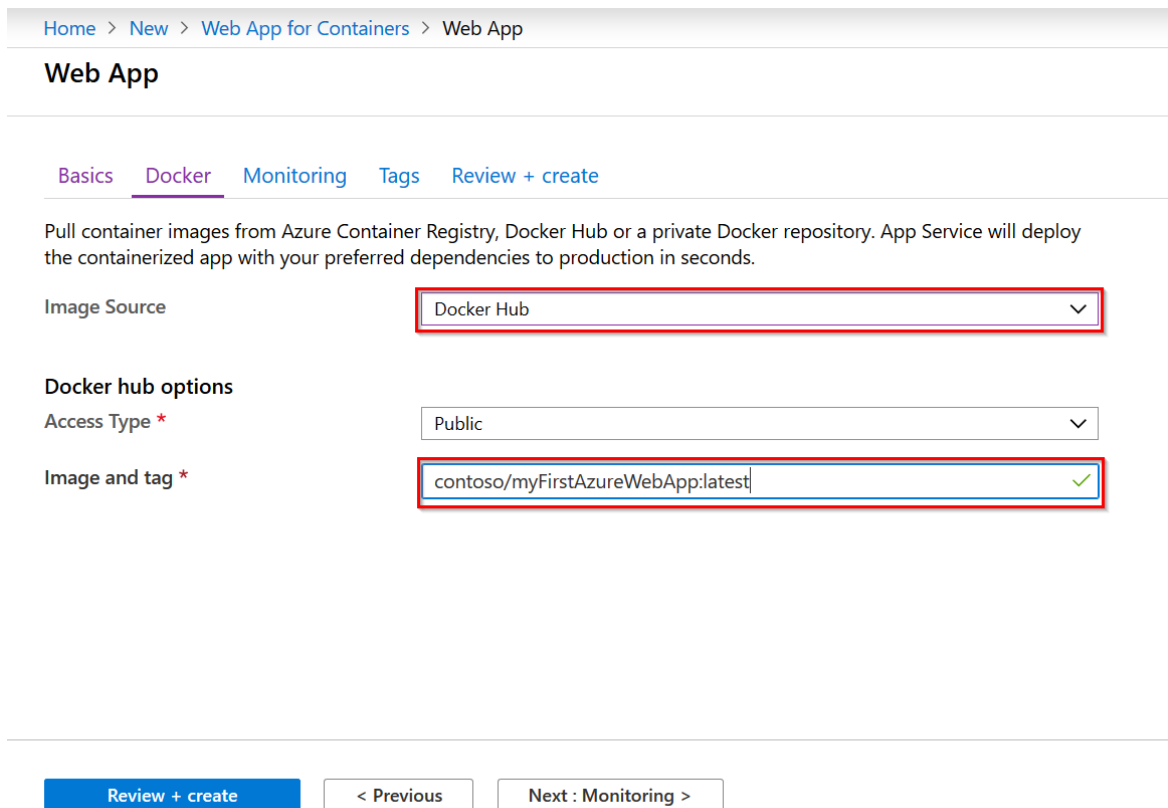
5. Copy this repository name for later.

Create a Windows container app

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



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



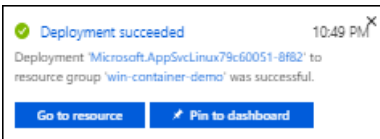
If you have a custom image elsewhere for your web application, such as in [Azure Container Registry](#) or in

any other private repository, you can configure it here.

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

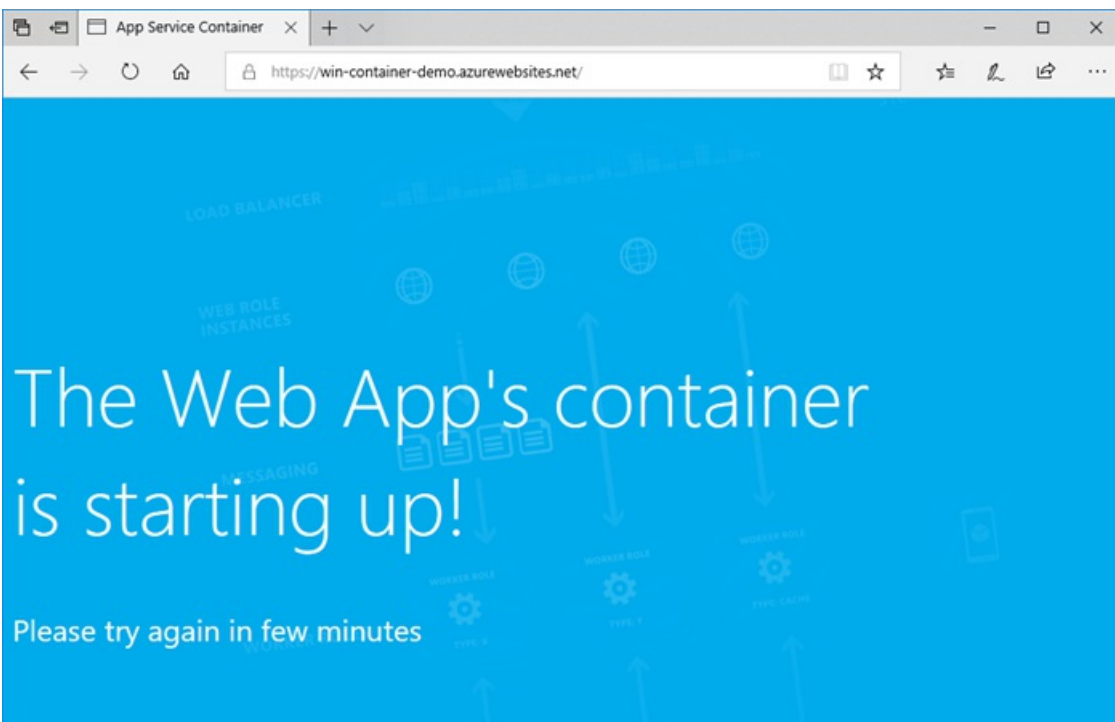
Browse to the container app

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

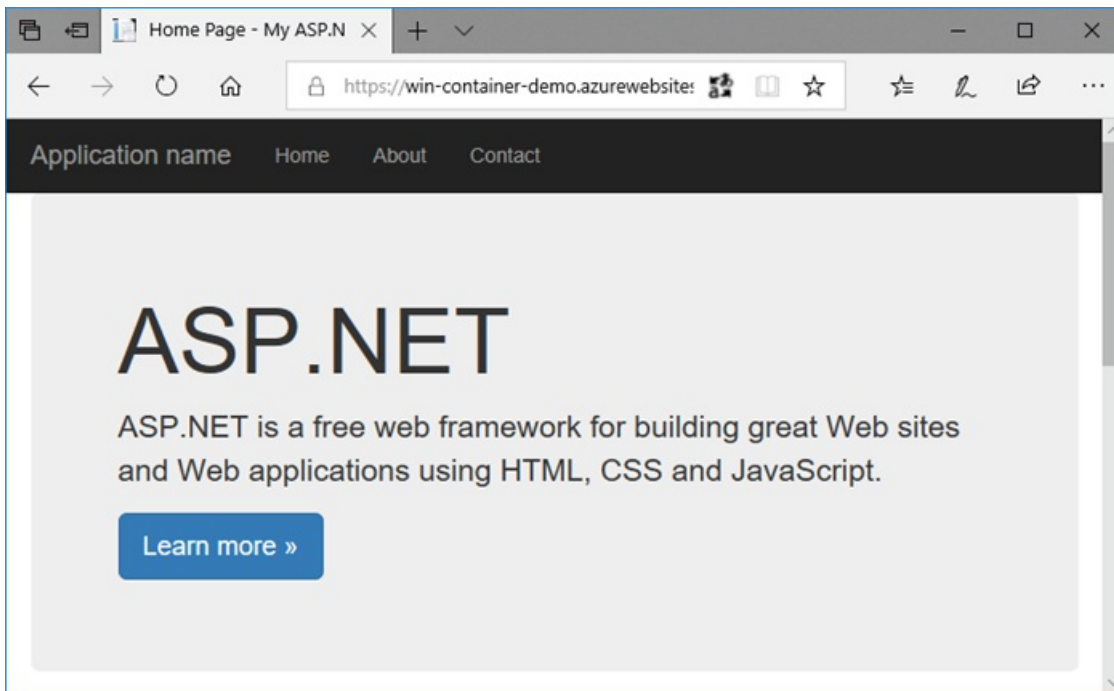


1. Click **Go to resource**.
2. In the overview of this resource, follow the link next to **URL**.

A new browser page opens to the following page:



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



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

See container start-up logs

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

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

The streamed logs looks like this:

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

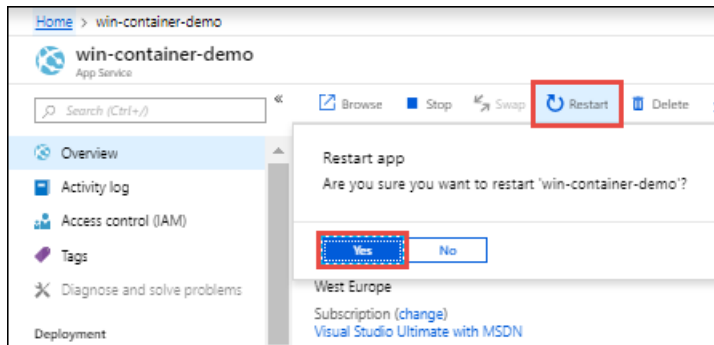
Update locally and redeploy

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

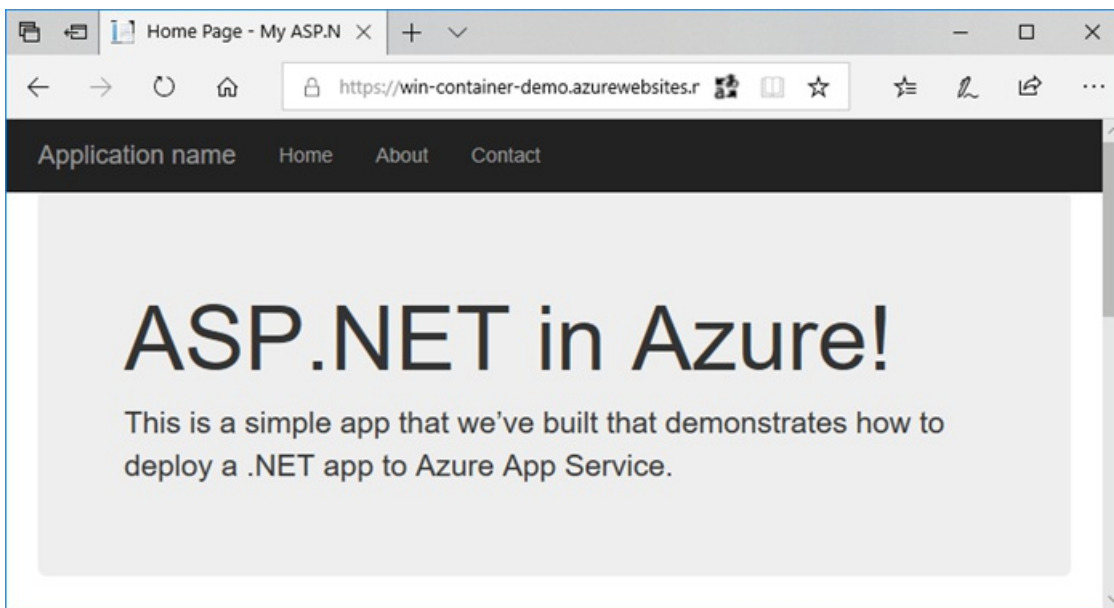
```
<div class="jumbotron">
  <h1>ASP.NET in Azure!</h1>
  <p class="lead">This is a simple app that we've built that demonstrates how to deploy a .NET app
to Azure App Service.</p>
</div>
```

3. To redeploy to Azure, right-click the **myfirstazurewebapp** project in **Solution Explorer** and choose **Publish**.
4. On the publish page, select **Publish** and wait for publishing to complete.

- To tell App Service to pull in the new image from Docker Hub, restart the app. Back in the app page in the portal, click **Restart** > **Yes**.



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



Next steps

[Migrate to Windows container in Azure](#)

Or, check out other resources:

[Configure custom container](#)

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

Prerequisites

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

Create an image

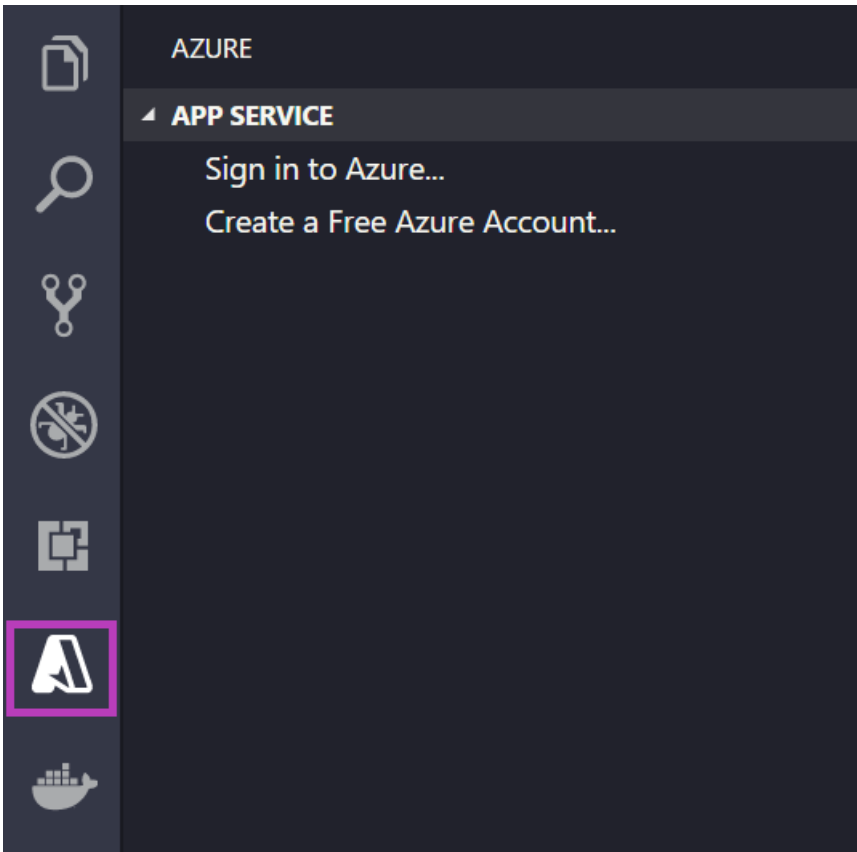
To complete this quickstart, you will need a suitable web app image stored in an [Azure Container Registry](#). Follow the instructions in [Quickstart: Create a private container registry using the Azure portal](#), but use the `mcr.microsoft.com/azuredocs/go` image instead of the `hello-world` image.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

IMPORTANT

Be sure to set the **Admin User** option to **Enable** when you create the container registry. You can also set it from the **Access keys** section of your registry page in the Azure portal. This setting is required for App Service access.

Sign in

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



Check prerequisites

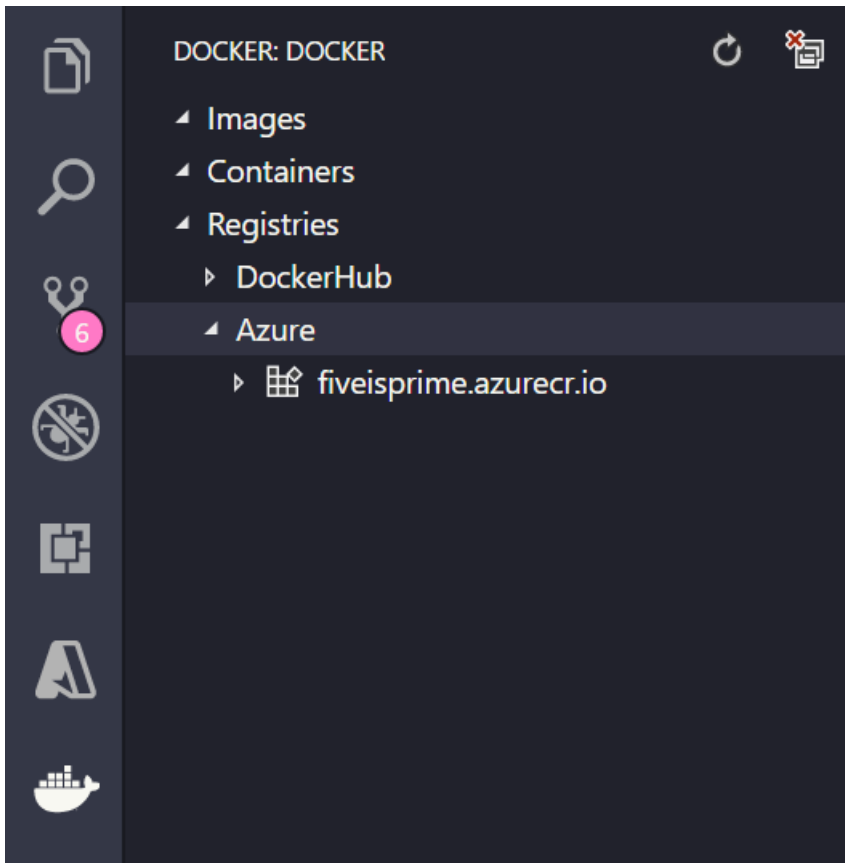
Now you can check whether you have all the prerequisites installed and configured properly.

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

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

```
docker --version
```

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



Deploy the image to Azure App Service

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

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

From here, follow the prompts to choose a subscription, a globally unique app name, a Resource Group, and an App Service Plan. Choose **B1 Basic** for the pricing tier, and a region.

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

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

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

Browse the website

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

[I ran into an issue](#)

Next steps

Congratulations, you've successfully completed this quickstart!

Next, check out the other Azure extensions.

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

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

Check out other resources:

[Configure custom container](#)

CLI samples for Azure App Service

6/23/2021 • 2 minutes to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure CLI.

SCRIPT	DESCRIPTION
Create app	
Create an app and deploy files with FTP	Creates an App Service app and deploys a file to it using FTP.
Create an app and deploy code from GitHub	Creates an App Service app and deploys code from a public GitHub repository.
Create an app with continuous deployment from GitHub	Creates an App Service app with continuous publishing from a GitHub repository you own.
Create an app and deploy code from a local Git repository	Creates an App Service app and configures code push from a local Git repository.
Create an app and deploy code to a staging environment	Creates an App Service app with a deployment slot for staging code changes.
Create an ASP.NET Core app in a Docker container	Creates an App Service app on Linux and loads a Docker image from Docker Hub.
Create an app and expose it with a Private Endpoint	Creates an App Service app and a Private Endpoint
Configure app	
Map a custom domain to an app	Creates an App Service app and maps a custom domain name to it.
Bind a custom TLS/SSL certificate to an app	Creates an App Service app and binds the TLS/SSL certificate of a custom domain name to it.
Scale app	
Scale an app manually	Creates an App Service app and scales it across 2 instances.
Scale an app worldwide with a high-availability architecture	Creates two App Service apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
Protect app	
Integrate with Azure Application Gateway	Creates an App Service app and integrates it with Application Gateway using service endpoint and access restrictions.
Connect app to resources	

SCRIPT	DESCRIPTION
Connect an app to a SQL Database	Creates an App Service app and a database in Azure SQL Database, then adds the database connection string to the app settings.
Connect an app to a storage account	Creates an App Service app and a storage account, then adds the storage connection string to the app settings.
Connect an app to an Azure Cache for Redis	Creates an App Service app and an Azure Cache for Redis, then adds the redis connection details to the app settings.)
Connect an app to Cosmos DB	Creates an App Service app and a Cosmos DB, then adds the Cosmos DB connection details to the app settings.
Backup and restore app	
Backup an app	Creates an App Service app and creates a one-time backup for it.
Create a scheduled backup for an app	Creates an App Service app and creates a scheduled backup for it.
Restores an app from a backup	Restores an App Service app from a backup.
Monitor app	
Monitor an app with web server logs	Creates an App Service app, enables logging for it, and downloads the logs to your local machine.

PowerShell samples for Azure App Service

11/2/2020 • 2 minutes to read • [Edit Online](#)

The following table includes links to PowerShell scripts built using the Azure PowerShell.

SCRIPT	DESCRIPTION
Create app	
Create an app with deployment from GitHub	Creates an App Service app that pulls code from GitHub.
Create an app with continuous deployment from GitHub	Creates an App Service app that continuously deploys code from GitHub.
Create an app and deploy code with FTP	Creates an App Service app and upload files from a local directory using FTP.
Create an app and deploy code from a local Git repository	Creates an App Service app and configures code push from a local Git repository.
Create an app and deploy code to a staging environment	Creates an App Service app with a deployment slot for staging code changes.
Create an app and expose your app with a Private Endpoint	Creates an App Service app with a Private Endpoint.
Configure app	
Map a custom domain to an app	Creates an App Service app and maps a custom domain name to it.
Bind a custom TLS/SSL certificate to an app	Creates an App Service app and binds the TLS/SSL certificate of a custom domain name to it.
Scale app	
Scale an app manually	Creates an App Service app and scales it across 2 instances.
Scale an app worldwide with a high-availability architecture	Creates two App Service apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
Connect app to resources	
Connect an app to a SQL Database	Creates an App Service app and a database in Azure SQL Database, then adds the database connection string to the app settings.
Connect an app to a storage account	Creates an App Service app and a storage account, then adds the storage connection string to the app settings.
Back up and restore app	

SCRIPT	DESCRIPTION
Back up an app	Creates an App Service app and creates a one-time backup for it.
Create a scheduled backup for an app	Creates an App Service app and creates a scheduled backup for it.
Delete a backup for an app	Deletes an existing backup for an app.
Restore an app from backup	Restores an app from a previously completed backup.
Restore a backup across subscriptions	Restores a web app from a backup in another subscription.
Monitor app	
Monitor an app with web server logs	Creates an App Service app, enables logging for it, and downloads the logs to your local machine.

What is Azure Cost Management + Billing?

3/5/2021 • 6 minutes to read • [Edit Online](#)

By using the Microsoft cloud, you can significantly improve the technical performance of your business workloads. It can also reduce your costs and the overhead required to manage organizational assets. However, the business opportunity creates a risk because of the potential for waste and inefficiencies that are introduced into your cloud deployments. Azure Cost Management + Billing is a suite of tools provided by Microsoft that help you analyze, manage, and optimize the costs of your workloads. Using the suite helps ensure that your organization is taking advantage of the benefits provided by the cloud.

You can think of your Azure workloads like the lights in your home. When you leave to go out for the day, are you leaving the lights on? Could you use different bulbs that are more efficient to help reduce your monthly energy bill? Do you have more lights in one room than are needed? You can use Azure Cost Management + Billing to apply a similar thought process to the workloads used by your organization.

With Azure products and services, you only pay for what you use. As you create and use Azure resources, you're charged for the resources. Because of the deployment ease for new resources, the costs of your workloads can jump significantly without proper analysis and monitoring. You use Azure Cost Management + Billing features to:

- Conduct billing administrative tasks such as paying your bill
- Manage billing access to costs
- Download cost and usage data that was used to generate your monthly invoice
- Proactively apply data analysis to your costs
- Set spending thresholds
- Identify opportunities for workload changes that can optimize your spending

To learn more about how to approach cost management as an organization, take a look at the [Azure Cost Management best practices](#) article.



Understand Azure Billing

Azure Billing features are used to review your invoiced costs and manage access to billing information. In larger organizations, procurement and finance teams usually conduct billing tasks.

A billing account is created when you sign up to use Azure. You use your billing account to manage your invoices, payments, and track costs. You can have access to multiple billing accounts. For example, you might have signed up for Azure for your personal projects. So, you might have an individual Azure subscription with a billing account. You could also have access through your organization's Enterprise Agreement or Microsoft Customer Agreement. For each scenario, you would have a separate billing account.

Billing accounts

The Azure portal currently supports the following types of billing accounts:

- **Microsoft Online Services Program:** An individual billing account for a Microsoft Online Services Program is created when you sign up for Azure through the Azure website. For example, when you sign up for an [Azure Free Account](#), account with pay-as-you-go rates or as a Visual studio subscriber.
- **Enterprise Agreement:** A billing account for an Enterprise Agreement is created when your organization signs an Enterprise Agreement (EA) to use Azure.
- **Microsoft Customer Agreement:** A billing account for a Microsoft Customer Agreement is created when your organization works with a Microsoft representative to sign a Microsoft Customer Agreement. Some customers in select regions, who sign up through the Azure website for an account with pay-as-you-go rates or upgrade their [Azure Free Account](#) may have a billing account for a Microsoft Customer Agreement as well.

Understand Azure Cost Management

Although related, billing differs from cost management. Billing is the process of invoicing customers for goods or services and managing the commercial relationship.

Cost Management shows organizational cost and usage patterns with advanced analytics. Reports in Cost Management show the usage-based costs consumed by Azure services and third-party Marketplace offerings. Costs are based on negotiated prices and factor in reservation and Azure Hybrid Benefit discounts. Collectively, the reports show your internal and external costs for usage and Azure Marketplace charges. Other charges, such as reservation purchases, support, and taxes aren't yet shown in reports. The reports help you understand your spending and resource use and can help find spending anomalies. Predictive analytics are also available. Cost Management uses Azure management groups, budgets, and recommendations to show clearly how your expenses are organized and how you might reduce costs.

You can use the Azure portal or various APIs for export automation to integrate cost data with external systems and processes. Automated billing data export and scheduled reports are also available.

Watch the Azure Cost Management overview video for a quick overview about how Azure Cost Management can help you save money in Azure. To watch other videos, visit the [Cost Management YouTube channel](#).

Plan and control expenses

The ways that Cost Management help you plan for and control your costs include: Cost analysis, budgets, recommendations, and exporting cost management data.

You use cost analysis to explore and analyze your organizational costs. You can view aggregated costs by organization to understand where costs are accrued and to identify spending trends. And you can see accumulated costs over time to estimate monthly, quarterly, or even yearly cost trends against a budget.

Budgets help you plan for and meet financial accountability in your organization. They help prevent cost thresholds or limits from being surpassed. Budgets can also help you inform others about their spending to proactively manage costs. And with them, you can see how spending progresses over time.

Recommendations show how you can optimize and improve efficiency by identifying idle and underutilized resources. Or, they can show less expensive resource options. When you act on the recommendations, you change the way you use your resources to save money. To act, you first view cost optimization recommendations to view potential usage inefficiencies. Next, you act on a recommendation to modify your Azure resource use to a more cost-effective option. Then you verify the action to make sure that the change you make is successful.

If you use external systems to access or review cost management data, you can easily export the data from Azure. And you can set a daily scheduled export in CSV format and store the data files in Azure storage. Then, you can access the data from your external system.

Cloudyn deprecation

Cloudyn is an Azure service related to Cost Management that is being deprecated by the end of 2020. Existing Cloudyn features are being integrated directly into the Azure portal wherever possible. No new customers are being onboarded at this time, but support will remain for the product until it is fully deprecated.

Additional Azure tools

Azure has other tools that aren't a part of the Azure Cost Management + Billing feature set. However, they play an important role in the cost management process. To learn more about these tools, see the following links.

- [Azure Pricing Calculator](#) - Use this tool to estimate your up-front cloud costs.
- [Azure Migrate](#) - Assess your current datacenter workload for insights about what's needed from an Azure replacement solution.
- [Azure Advisor](#) - Identify unused VMs and receive recommendations about Azure reserved instance

purchases.

- [Azure Hybrid Benefit](#) - Use your current on-premises Windows Server or SQL Server licenses for VMs in Azure to save.

Next steps

Now that you're familiar with Cost Management + Billing, the next step is to start using the service.

- Start using Azure Cost Management to [analyze costs](#).
- You can also read more about [Azure Cost Management best practices](#).

App Service overview

4/28/2021 • 5 minutes to read • [Edit Online](#)

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and [Linux](#)-based environments.

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

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

Why use App Service?

Here are some key features of App Service:

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

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

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

App Service on Linux

App Service can also host web apps natively on Linux for supported application stacks. It can also run custom Linux containers (also known as Web App for Containers).

Built-in languages and frameworks

App Service on Linux supports a number of language specific built-in images. Just deploy your code. Supported languages include: Node.js, Java (JRE 8 & JRE 11), PHP, Python, .NET Core, and Ruby. Run

```
az webapp list-runtimes --linux
```

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

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

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

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

Limitations

- App Service on Linux is not supported on [Shared](#) pricing tier.
- You can't mix Windows and Linux apps in the same App Service plan.
- Historically, you can't mix Windows and Linux apps in the same resource group. However, all resource groups created on or after January 21, 2021 do support this scenario. For resource groups created before January 21, 2021, the ability to add mixed platform deployments will be rolled out across Azure regions (including National cloud regions) soon.
- The Azure portal shows only features that currently work for Linux apps. As features are enabled, they're activated on the portal.
- When deployed to built-in images, your code and content are allocated a storage volume for web content, backed by Azure Storage. The disk latency of this volume is higher and more variable than the latency of the container filesystem. Apps that require heavy read-only access to content files may benefit from the custom container option, which places files in the container filesystem instead of on the content volume.

Next steps

Create your first web app.

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

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

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

Ruby (on Linux)

Nodejs (on Windows or Linux)

Java (on Windows or Linux)

Python (on Linux)

HTML (on Windows or Linux)

Custom container (Windows or Linux)

Linux virtual machines in Azure

3/10/2021 • 5 minutes to read • [Edit Online](#)

Azure Virtual Machines (VM) is one of several types of [on-demand, scalable computing resources](#) that Azure offers. Typically, you choose a VM when you need more control over the computing environment than the other choices offer. This article gives you information about what you should consider before you create a VM, how you create it, and how you manage it.

An Azure VM gives you the flexibility of virtualization without having to buy and maintain the physical hardware that runs it. However, you still need to maintain the VM by performing tasks, such as configuring, patching, and installing the software that runs on it.

Azure virtual machines can be used in various ways. Some examples are:

- **Development and test** – Azure VMs offer a quick and easy way to create a computer with specific configurations required to code and test an application.
- **Applications in the cloud** – Because demand for your application can fluctuate, it might make economic sense to run it on a VM in Azure. You pay for extra VMs when you need them and shut them down when you don't.
- **Extended datacenter** – Virtual machines in an Azure virtual network can easily be connected to your organization's network.

The number of VMs that your application uses can scale up and out to whatever is required to meet your needs.

What do I need to think about before creating a VM?

There are always a multitude of [design considerations](#) when you build out an application infrastructure in Azure. These aspects of a VM are important to think about before you start:

- The names of your application resources
- The location where the resources are stored
- The size of the VM
- The maximum number of VMs that can be created
- The operating system that the VM runs
- The configuration of the VM after it starts
- The related resources that the VM needs

Locations

All resources created in Azure are distributed across multiple [geographical regions](#) around the world. Usually, the region is called **location** when you create a VM. For a VM, the location specifies where the virtual hard disks are stored.

This table shows some of the ways you can get a list of available locations.

METHOD	DESCRIPTION
Azure portal	Select a location from the list when you create a VM.
Azure PowerShell	Use the Get-AzLocation command.

METHOD	DESCRIPTION
REST API	Use the List locations operation.
Azure CLI	Use the az account list-locations operation.

Availability

Azure announced an industry leading single instance virtual machine Service Level Agreement of 99.9% provided you deploy the VM with premium storage for all disks. In order for your deployment to qualify for the standard 99.95% VM Service Level Agreement, you still need to deploy two or more VMs running your workload inside of an availability set. An availability set ensures that your VMs are distributed across multiple fault domains in the Azure data centers as well as deployed onto hosts with different maintenance windows. The full [Azure SLA](#) explains the guaranteed availability of Azure as a whole.

VM Size

The [size](#) of the VM that you use is determined by the workload that you want to run. The size that you choose then determines factors such as processing power, memory, and storage capacity. Azure offers a wide variety of sizes to support many types of uses.

Azure charges an [hourly price](#) based on the VM's size and operating system. For partial hours, Azure charges only for the minutes used. Storage is priced and charged separately.

VM Limits

Your subscription has default [quota limits](#) in place that could impact the deployment of many VMs for your project. The current limit on a per subscription basis is 20 VMs per region. Limits can be raised by [filing a support ticket requesting an increase](#)

Managed Disks

Managed Disks handles Azure Storage account creation and management in the background for you, and ensures that you do not have to worry about the scalability limits of the storage account. You specify the disk size and the performance tier (Standard or Premium), and Azure creates and manages the disk. As you add disks or scale the VM up and down, you don't have to worry about the storage being used. If you're creating new VMs, [use the Azure CLI](#) or the Azure portal to create VMs with Managed OS and data disks. If you have VMs with unmanaged disks, you can [convert your VMs to be backed with Managed Disks](#).

You can also manage your custom images in one storage account per Azure region, and use them to create hundreds of VMs in the same subscription. For more information about Managed Disks, see the [Managed Disks Overview](#).

Distributions

Microsoft Azure supports running a number of popular Linux distributions provided and maintained by a number of partners. You can find available distributions in the Azure Marketplace. Microsoft actively works with various Linux communities to add even more flavors to the [Azure endorsed Linux Distros](#) list.

If your preferred Linux distro of choice is not currently present in the gallery, you can "Bring your own Linux" VM by [creating and uploading a Linux VHD in Azure](#).

Microsoft works closely with partners to ensure the images available are updated and optimized for an Azure runtime. For more information on Azure partner offers, see the following links:

- [Linux on Azure - Endorsed Distributions](#)
- [SUSE - Azure Marketplace - SUSE Linux Enterprise Server](#)
- [Red Hat - Azure Marketplace - Red Hat Enterprise Linux](#)
- [Canonical - Azure Marketplace - Ubuntu Server](#)
- [Debian - Azure Marketplace - Debian](#)
- [FreeBSD - Azure Marketplace - FreeBSD](#)
- [Flatcar - Azure Marketplace - Flatcar Container Linux](#)
- [RancherOS - Azure Marketplace - RancherOS](#)
- [Bitnami - Bitnami Library for Azure](#)
- [Mesosphere - Azure Marketplace - Mesosphere DC/OS on Azure](#)
- [Docker - Azure Marketplace - Docker images](#)
- [Jenkins - Azure Marketplace - CloudBees Jenkins Platform](#)

Cloud-init

To achieve a proper DevOps culture, all infrastructure must be code. When all the infrastructure lives in code it can easily be recreated. Azure works with all the major automation tooling like Ansible, Chef, SaltStack, and Puppet. Azure also has its own tooling for automation:

- [Azure Templates](#)
- [Azure VMaccess](#)

Azure supports for [cloud-init](#) across most Linux Distros that support it. We are actively working with our endorsed Linux distro partners in order to have cloud-init enabled images available in the Azure marketplace. These images will make your cloud-init deployments and configurations work seamlessly with VMs and virtual machine scale sets.

- [Using cloud-init on Azure Linux VMs](#)

Storage

- [Introduction to Microsoft Azure Storage](#)
- [Add a disk to a Linux VM using the azure-cli](#)
- [How to attach a data disk to a Linux VM in the Azure portal](#)

Networking

- [Virtual Network Overview](#)
- [IP addresses in Azure](#)
- [Opening ports to a Linux VM in Azure](#)
- [Create a Fully Qualified Domain Name in the Azure portal](#)

Data residency

In Azure, the feature to enable storing customer data in a single region is currently only available in the Southeast Asia Region (Singapore) of the Asia Pacific Geo and Brazil South (Sao Paulo State) Region of Brazil Geo. For all other regions, customer data is stored in Geo. For more information, see [Trust Center](#).

Next steps

Create your first VM!

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

Windows virtual machines in Azure

4/11/2021 • 5 minutes to read • [Edit Online](#)

Azure Virtual Machines (VM) is one of several types of [on-demand, scalable computing resources](#) that Azure offers. Typically, you choose a VM when you need more control over the computing environment than the other choices offer. This article gives you information about what you should consider before you create a VM, how you create it, and how you manage it.

An Azure VM gives you the flexibility of virtualization without having to buy and maintain the physical hardware that runs it. However, you still need to maintain the VM by performing tasks, such as configuring, patching, and installing the software that runs on it.

Azure virtual machines can be used in various ways. Some examples are:

- **Development and test** – Azure VMs offer a quick and easy way to create a computer with specific configurations required to code and test an application.
- **Applications in the cloud** – Because demand for your application can fluctuate, it might make economic sense to run it on a VM in Azure. You pay for extra VMs when you need them and shut them down when you don't.
- **Extended datacenter** – Virtual machines in an Azure virtual network can easily be connected to your organization's network.

The number of VMs that your application uses can scale up and out to whatever is required to meet your needs.

What do I need to think about before creating a VM?

There are always a multitude of [design considerations](#) when you build out an application infrastructure in Azure. These aspects of a VM are important to think about before you start:

- The names of your application resources
- The location where the resources are stored
- The size of the VM
- The maximum number of VMs that can be created
- The operating system that the VM runs
- The configuration of the VM after it starts
- The related resources that the VM needs

Locations

All resources created in Azure are distributed across multiple [geographical regions](#) around the world. Usually, the region is called **location** when you create a VM. For a VM, the location specifies where the virtual hard disks are stored.

This table shows some of the ways you can get a list of available locations.

METHOD	DESCRIPTION
Azure portal	Select a location from the list when you create a VM.
Azure PowerShell	Use the Get-AzLocation command.

METHOD	DESCRIPTION
REST API	Use the List locations operation.
Azure CLI	Use the az account list-locations operation.

Availability

Azure announced an industry leading single instance virtual machine Service Level Agreement of 99.9% provided you deploy the VM with premium storage for all disks. In order for your deployment to qualify for the standard 99.95% VM Service Level Agreement, you still need to deploy two or more VMs running your workload inside of an availability set. An availability set ensures that your VMs are distributed across multiple fault domains in the Azure data centers as well as deployed onto hosts with different maintenance windows. The full [Azure SLA](#) explains the guaranteed availability of Azure as a whole.

VM size

The [size](#) of the VM that you use is determined by the workload that you want to run. The size that you choose then determines factors such as processing power, memory, and storage capacity. Azure offers a wide variety of sizes to support many types of uses.

Azure charges an [hourly price](#) based on the VM's size and operating system. For partial hours, Azure charges only for the minutes used. Storage is priced and charged separately.

VM Limits

Your subscription has default [quota limits](#) in place that could impact the deployment of many VMs for your project. The current limit on a per subscription basis is 20 VMs per region. Limits can be raised by [filing a support ticket requesting an increase](#)

Operating system disks and images

Virtual machines use [virtual hard disks \(VHDs\)](#) to store their operating system (OS) and data. VHDs are also used for the images you can choose from to install an OS.

Azure provides many [marketplace images](#) to use with various versions and types of Windows Server operating systems. Marketplace images are identified by image publisher, offer, sku, and version (typically version is specified as latest). Only 64-bit operating systems are supported. For more information on the supported guest operating systems, roles, and features, see [Microsoft server software support for Microsoft Azure virtual machines](#).

This table shows some ways that you can find the information for an image.

METHOD	DESCRIPTION
Azure portal	The values are automatically specified for you when you select an image to use.
Azure PowerShell	Get-AzVMImagePublisher -Location <i>location</i> Get-AzVMImageOffer -Location <i>location</i> -Publisher <i>publisherName</i> Get-AzVMImageSku -Location <i>location</i> -Publisher <i>publisherName</i> -Offer <i>offerName</i>

METHOD	DESCRIPTION
REST APIs	List image publishers List image offers List image skus
Azure CLI	<pre>az vm image list-publishers --location <i>location</i> az vm image list-offers --location <i>location</i> --publisher <i>publisherName</i> az vm image list-skus --location <i>location</i> --publisher <i>publisherName</i> --offer <i>offerName</i></pre>

You can choose to [upload and use your own image](#) and when you do, the publisher name, offer, and sku aren't used.

Extensions

VM [extensions](#) give your VM additional capabilities through post deployment configuration and automated tasks.

These common tasks can be accomplished using extensions:

- **Run custom scripts** – The [Custom Script Extension](#) helps you configure workloads on the VM by running your script when the VM is provisioned.
- **Deploy and manage configurations** – The [PowerShell Desired State Configuration \(DSC\) Extension](#) helps you set up DSC on a VM to manage configurations and environments.
- **Collect diagnostics data** – The [Azure Diagnostics Extension](#) helps you configure the VM to collect diagnostics data that can be used to monitor the health of your application.

Related resources

The resources in this table are used by the VM and need to exist or be created when the VM is created.

RESOURCE	REQUIRED	DESCRIPTION
Resource group	Yes	The VM must be contained in a resource group.
Storage account	Yes	The VM needs the storage account to store its virtual hard disks.
Virtual network	Yes	The VM must be a member of a virtual network.
Public IP address	No	The VM can have a public IP address assigned to it to remotely access it.
Network interface	Yes	The VM needs the network interface to communicate in the network.
Data disks	No	The VM can include data disks to expand storage capabilities.

Data residency

In Azure, the feature to enable storing customer data in a single region is currently only available in the Southeast Asia Region (Singapore) of the Asia Pacific Geo and Brazil South (Sao Paulo State) Region of Brazil

Geo. For all other regions, customer data is stored in Geo. For more information, see [Trust Center](#).

Next steps

Create your first VM!

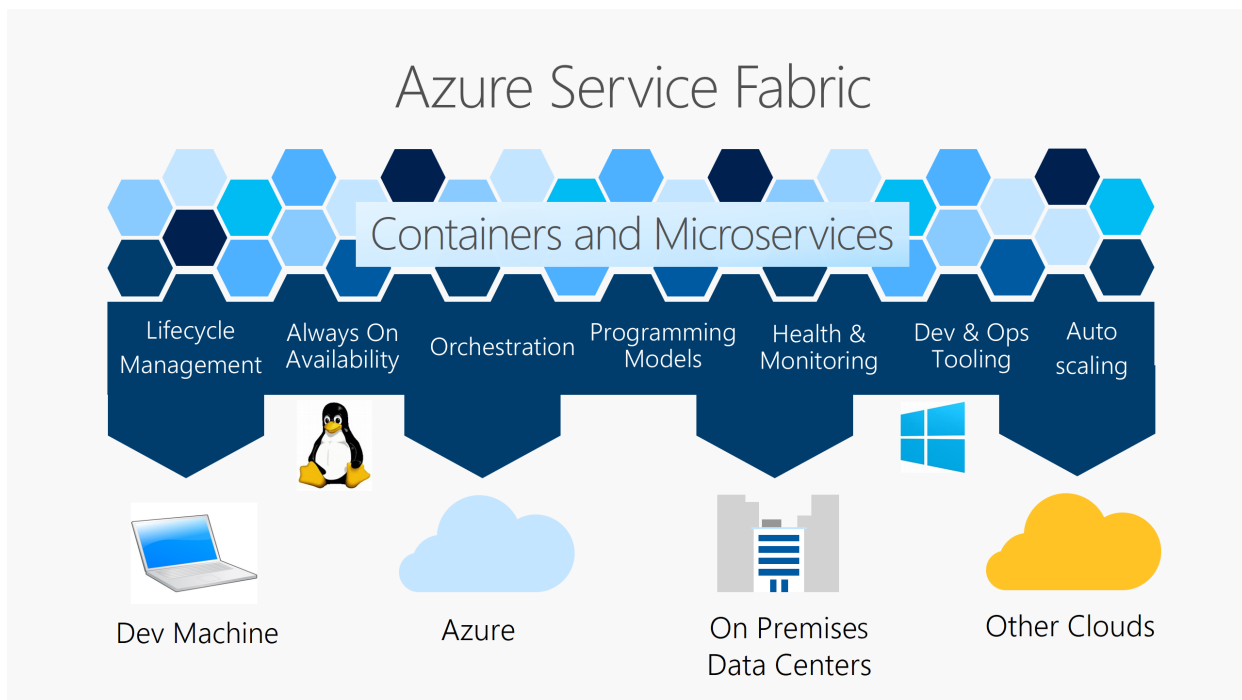
- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Overview of Azure Service Fabric

3/5/2021 • 2 minutes to read • [Edit Online](#)

Azure Service Fabric is a [distributed systems platform](#) that makes it easy to package, deploy, and manage scalable and reliable microservices and containers. Service Fabric also addresses the significant challenges in [developing and managing](#) cloud native applications.

A key differentiator of Service Fabric is its strong focus on building stateful services. You can use the Service Fabric [programming model](#) or run containerized stateful services written in any language or code. You can create [Service Fabric clusters anywhere](#), including Windows Server and Linux on premises and other public clouds, in addition to Azure.



Service Fabric powers many Microsoft services today, including Azure SQL Database, Azure Cosmos DB, Cortana, Microsoft Power BI, Microsoft Intune, Azure Event Hubs, Azure IoT Hub, Dynamics 365, Skype for Business, and many core Azure services.

Container orchestration

Service Fabric is Microsoft's [container orchestrator](#) for deploying and managing microservices across a cluster of machines, benefiting from the lessons learned running Microsoft services at massive scale. Service Fabric can deploy applications in seconds, at high density with hundreds or thousands of applications or containers per machine. With Service Fabric, you can mix both services in processes and services in containers in the same application.

[Learn more about Service Fabric](#) core concepts, programming models, application lifecycle, testing, clusters, and health monitoring.

Stateless and stateful microservices

Service Fabric provides a sophisticated, lightweight runtime that supports stateless and stateful microservices. A key differentiator of Service Fabric is its robust support for building stateful services, either with Service Fabric [built-in programming models](#) or containerized stateful services.

Learn more about [application scenarios](#) that benefit from Service Fabric stateful services.

Application lifecycle management

Service Fabric provides support for the full application lifecycle and CI/CD of cloud applications including containers: development through deployment, daily monitoring, management, and maintenance, to eventual decommissioning. Service Fabric is integrated with CI/CD tools such as [Azure Pipelines](#), [Jenkins](#), and [Octopus Deploy](#) and can be used with any other popular CI/CD tool.

For more information about application lifecycle management, read [Application lifecycle](#). For deploying existing applications to Service Fabric, see [Deploy a guest executable](#).

Any OS, any cloud

You can create clusters for Service Fabric in many environments, including [Azure or on premises](#), on [Windows Server or Linux](#). You can even create clusters on other public clouds. The development environment in the Service Fabric SDK is identical to the production environment, with no emulators involved. In other words, what runs on your local development cluster is what deploys to your clusters in other environments.

For [Windows development](#), the Service Fabric .NET SDK is integrated with Visual Studio and PowerShell. For [Linux development](#), the Service Fabric Java SDK is integrated with Eclipse, and Yeoman is used to generate templates for Java, .NET Core, and container applications.

Compliance

Azure Service Fabric Resource Provider is available in all Azure regions and is compliant with all Azure compliance certifications, including: SOC, ISO, PCI DSS, HIPAA, and GDPR. For a complete list, see [Microsoft Compliance Offerings](#).

Next steps

Create and deploy your first application on Azure Service Fabric:

[Service Fabric quickstart](#)

Service Fabric application scenarios

3/24/2021 • 4 minutes to read • [Edit Online](#)

Azure Service Fabric offers a reliable and flexible platform where you can write and run many types of business applications and services. These applications and microservices can be stateless or stateful, and they're resource-balanced across virtual machines to maximize efficiency.

The unique architecture of Service Fabric enables you to perform near real-time data analysis, in-memory computation, parallel transactions, and event processing in your applications. You can easily scale your applications in or out depending on your changing resource requirements.

For design guidance on building applications, read [Microservices architecture on Azure Service Fabric](#) and [Best practices for application design using Service Fabric](#).

Consider using the Service Fabric platform for the following types of applications:

- **Data gathering, processing, and IoT:** Service Fabric handles large scale and has low latency through its stateful services. It can help process data on millions of devices where the data for the device and the computation are colocated.

Customers who have built IoT services by using Service Fabric include [Honeywell](#), [PCL Construction](#), [Crestron](#), [BMW](#), [Schneider Electric](#), and [Mesh Systems](#).

- **Gaming and session-based interactive applications:** Service Fabric is useful if your application requires low-latency reads and writes, such as in online gaming or instant messaging. Service Fabric enables you to build these interactive, stateful applications without having to create a separate store or cache. Visit [Azure gaming solutions](#) for design guidance on [using Service Fabric in gaming services](#).

Customers who have built gaming services include [Next Games](#) and [Digamore](#). Customers who have built interactive sessions include [Honeywell with Hololens](#).

- **Data analytics and workflow processing:** Applications that must reliably process events or streams of data benefit from the optimized reads and writes in Service Fabric. Service Fabric also supports application processing pipelines, where results must be reliable and passed on to the next processing stage without any loss. These pipelines include transactional and financial systems, where data consistency and computation guarantees are essential.

Customers who have built business workflow services include [Zeiss Group](#), [Quorum Business Solutions](#), and [Société General](#).

- **Computation on data:** Service Fabric enables you to build stateful applications that do intensive data computation. Service Fabric allows the colocation of processing (computation) and data in applications.

Normally, when your application requires access to data, network latency associated with an external data cache or storage tier limits the computation time. Stateful Service Fabric services eliminate that latency, enabling more optimized reads and writes.

For example, consider an application that performs near real-time recommendation selections for customers, with a round-trip time requirement of less than 100 milliseconds. The latency and performance characteristics of Service Fabric services provide a responsive experience to the user, compared with the standard implementation model of having to fetch the necessary data from remote storage. The system is more responsive because the computation of recommendation selection is colocated with the data and rules.

Customers who have built computation services include [Solidsoft Reply](#) and [Infosupport](#).

- **Highly available services:** Service Fabric provides fast failover by creating multiple secondary service replicas. If a node, process, or individual service goes down due to hardware or other failure, one of the secondary replicas is promoted to a primary replica with minimal loss of service.
- **Scalable services:** Individual services can be partitioned, allowing for state to be scaled out across the cluster. Individual services can also be created and removed on the fly. You can scale out services from a few instances on a few nodes to thousands of instances on many nodes, and then scale them in again as needed. You can use Service Fabric to build these services and manage their complete life cycles.

Application design case studies

Case studies that show how Service Fabric is used to design applications are published on the [Customer stories](#) and [Microservices in Azure](#) sites.

Designing applications composed of stateless and stateful microservices

Building applications with Azure Cloud Services worker roles is an example of a stateless service. In contrast, stateful microservices maintain their authoritative state beyond the request and its response. This functionality provides high availability and consistency of the state through simple APIs that provide transactional guarantees backed by replication.

Stateful services in Service Fabric bring high availability to all types of applications, not just databases and other data stores. This is a natural progression. Applications have already moved from using purely relational databases for high availability to NoSQL databases. Now the applications themselves can have their "hot" state and data managed within them for additional performance gains without sacrificing reliability, consistency, or availability.

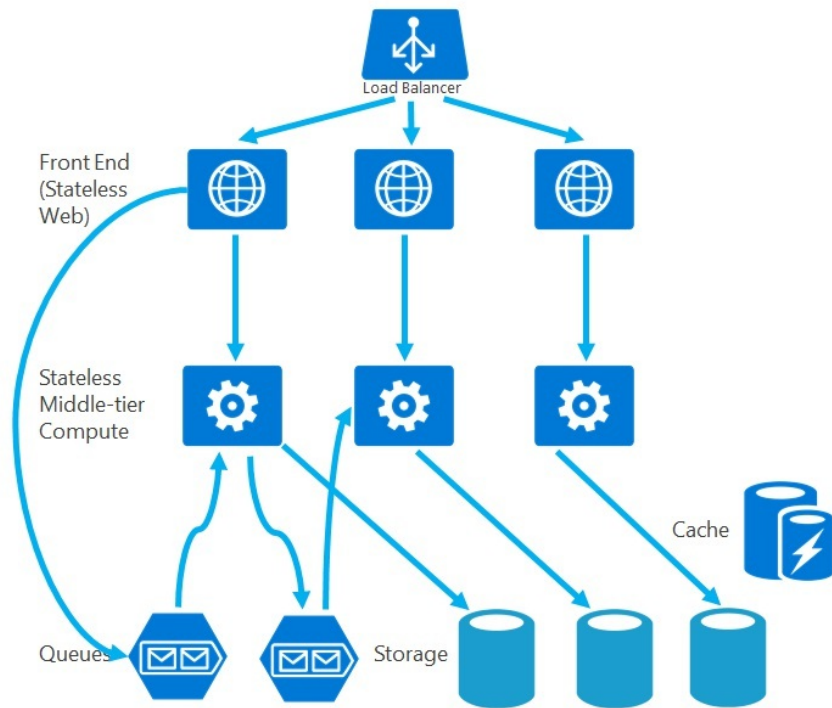
When you're building applications that consist of microservices, you typically have a combination of stateless web apps (like ASP.NET and Node.js) calling onto stateless and stateful business middle-tier services. The apps and services are all deployed in the same Service Fabric cluster through the Service Fabric deployment commands. Each of these services is independent with regard to scale, reliability, and resource usage. This independence improves agility and flexibility in development and life-cycle management.

Stateful microservices simplify application designs because they remove the need for the additional queues and caches that have traditionally been required to address the availability and latency requirements of purely stateless applications. Because stateful services have high availability and low latency, there are fewer details to manage in your application.

The following diagrams illustrate the differences between designing an application that's stateless and one that's stateful. By taking advantage of the [Reliable Services](#) and [Reliable Actors](#) programming models, stateful services reduce application complexity while achieving high throughput and low latency.

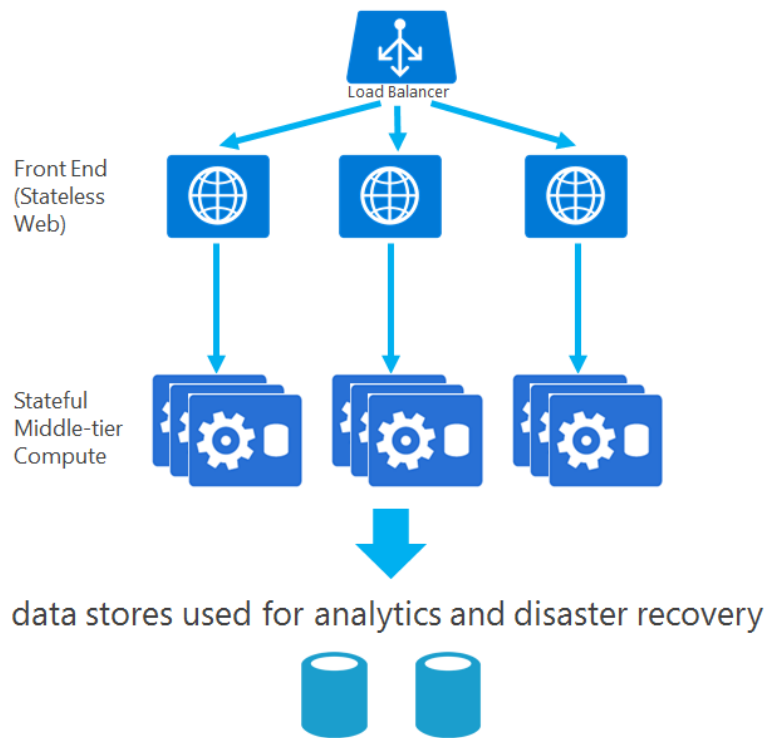
Here's an example application that uses stateless services:

- Scale with partitioned storage
- Increase reliability with queues
- Reduce read latency with caches
- Write your own lock managers for state consistency
- Many moving parts each managed differently



Here's an example application that uses stateful services:

- Application state lives in the compute tier
- Low Latency reads and writes
- Partitions are first class for scale-out
- Built in lock managers based on primary election
- Fewer moving parts



Next steps

- Get started building stateless and stateful services with the Service Fabric [Reliable Services](#) and [Reliable Actors](#) programming models.
- Visit the Azure Architecture Center for guidance on [building microservices on Azure](#).
- Go to [Azure Service Fabric application and cluster best practices](#) for application design guidance.
- See also:
 - [Understanding microservices](#)
 - [Define and manage service state](#)
 - [Availability of services](#)

- Scale services
- Partition services

How to create a Linux virtual machine with Azure Resource Manager templates

5/3/2021 • 4 minutes to read • [Edit Online](#)

Learn how to create a Linux virtual machine (VM) by using an Azure Resource Manager template and the Azure CLI from the Azure Cloud shell. To create a Windows virtual machine, see [Create a Windows virtual machine from a Resource Manager template](#).

An alternative is to deploy the template from the Azure portal. To open the template in the portal, select the **Deploy to Azure** button.



Templates overview

Azure Resource Manager templates are JSON files that define the infrastructure and configuration of your Azure solution. By using a template, you can repeatedly deploy your solution throughout its lifecycle and have confidence your resources are deployed in a consistent state. To learn more about the format of the template and how you construct it, see [Quickstart: Create and deploy Azure Resource Manager templates by using the Azure portal](#). To view the JSON syntax for resources types, see [Define resources in Azure Resource Manager templates](#).

Create a virtual machine

Creating an Azure virtual machine usually includes two steps:

1. Create a resource group. An Azure resource group is a logical container into which Azure resources are deployed and managed. A resource group must be created before a virtual machine.
2. Create a virtual machine.

The following example creates a VM from an [Azure Quickstart template](#). Only SSH authentication is allowed for this deployment. When prompted, provide the value of your own SSH public key, such as the contents of `~/.ssh/id_rsa.pub`. If you need to create an SSH key pair, see [How to create and use an SSH key pair for Linux VMs in Azure](#). Here is a copy of the template:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "projectName": {
      "type": "string",
      "metadata": {
        "description": "Specifies a name for generating resource names."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Specifies the location for all resources."
      }
    },
    "adminUsername": {
      "type": "string",
      "metadata": {
        "description": "Specifies the administrator username for the virtual machine."
      }
    }
  }
}
```

```

    "type": "string",
    "metadata": {
      "description": "Specifies a username for the Virtual Machine."
    }
  },
  "adminPublicKey": {
    "type": "string",
    "metadata": {
      "description": "Specifies the SSH rsa public key file as a string. Use \"ssh-keygen -t rsa -b 2048\"
to generate your SSH key pairs."
    }
  },
  "vmSize": {
    "type": "string",
    "defaultValue": "Standard_D2s_v3",
    "metadata": {
      "description": "description"
    }
  }
},
"variables": {
  "vNetName": "[concat(parameters('projectName'), '-vnet')]",
  "vNetAddressPrefixes": "10.0.0.0/16",
  "vNetSubnetName": "default",
  "vNetSubnetAddressPrefix": "10.0.0.0/24",
  "vmName": "[concat(parameters('projectName'), '-vm')]",
  "publicIPAddressName": "[concat(parameters('projectName'), '-ip')]",
  "networkInterfaceName": "[concat(parameters('projectName'), '-nic')]",
  "networkSecurityGroupName": "[concat(parameters('projectName'), '-nsg')]",
  "networkSecurityGroupName2": "[concat(variables('vNetSubnetName'), '-nsg')]"
},
"resources": [
  {
    "type": "Microsoft.Network/networkSecurityGroups",
    "apiVersion": "2020-05-01",
    "name": "[variables('networkSecurityGroupName')]",
    "location": "[parameters('location')]",
    "properties": {
      "securityRules": [
        {
          "name": "ssh_rule",
          "properties": {
            "description": "Locks inbound down to ssh default port 22.",
            "protocol": "Tcp",
            "sourcePortRange": "*",
            "destinationPortRange": "22",
            "sourceAddressPrefix": "*",
            "destinationAddressPrefix": "*",
            "access": "Allow",
            "priority": 123,
            "direction": "Inbound"
          }
        }
      ]
    }
  },
  {
    "type": "Microsoft.Network/publicIPAddresses",
    "apiVersion": "2020-05-01",
    "name": "[variables('publicIPAddressName')]",
    "location": "[parameters('location')]",
    "properties": {
      "publicIPAllocationMethod": "Dynamic"
    },
    "sku": {
      "name": "Basic"
    }
  }
],
{

```

```

"comments": "Simple Network Security Group for subnet [variables('vNetSubnetName')]",
"type": "Microsoft.Network/networkSecurityGroups",
"apiVersion": "2020-05-01",
"name": "[variables('networkSecurityGroupName2')]",
"location": "[parameters('location')]",
"properties": {
  "securityRules": [
    {
      "name": "default-allow-22",
      "properties": {
        "priority": 1000,
        "access": "Allow",
        "direction": "Inbound",
        "destinationPortRange": "22",
        "protocol": "Tcp",
        "sourceAddressPrefix": "*",
        "sourcePortRange": "*",
        "destinationAddressPrefix": "*"
      }
    }
  ]
},
{
  "type": "Microsoft.Network/virtualNetworks",
  "apiVersion": "2020-05-01",
  "name": "[variables('vNetName')]",
  "location": "[parameters('location')]",
  "dependsOn": [
    "[resourceId('Microsoft.Network/networkSecurityGroups', variables('networkSecurityGroupName2'))]"
  ],
  "properties": {
    "addressSpace": {
      "addressPrefixes": [
        "[variables('vNetAddressPrefixes')]"
      ]
    },
    "subnets": [
      {
        "name": "[variables('vNetSubnetName')]",
        "properties": {
          "addressPrefix": "[variables('vNetSubnetAddressPrefix')]",
          "networkSecurityGroup": {
            "id": "[resourceId('Microsoft.Network/networkSecurityGroups',
variables('networkSecurityGroupName2'))]"
          }
        }
      }
    ]
  }
},
{
  "type": "Microsoft.Network/networkInterfaces",
  "apiVersion": "2020-05-01",
  "name": "[variables('networkInterfaceName')]",
  "location": "[parameters('location')]",
  "dependsOn": [
    "[resourceId('Microsoft.Network/publicIPAddresses', variables('publicIPAddressName'))]",
    "[resourceId('Microsoft.Network/virtualNetworks', variables('vNetName'))]",
    "[resourceId('Microsoft.Network/networkSecurityGroups', variables('networkSecurityGroupName'))]"
  ],
  "properties": {
    "ipConfigurations": [
      {
        "name": "ipconfig1",
        "properties": {
          "privateIPAllocationMethod": "Dynamic",
          "publicIPAddress": {
            "id": "[resourceId('Microsoft.Network/publicIPAddresses',

```

```

variables('publicIPAddressName'))]"
    },
    "subnet": {
      "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets', variables('vNetName'),
variables('vNetSubnetName'))]"
    }
  }
}
],
},
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2019-12-01",
  "name": "[variables('vmName')]",
  "location": "[parameters('location')]",
  "dependsOn": [
    "[resourceId('Microsoft.Network/networkInterfaces', variables('networkInterfaceName'))]"
  ],
  "properties": {
    "hardwareProfile": {
      "vmSize": "[parameters('vmSize')]"
    },
    "osProfile": {
      "computerName": "[variables('vmName')]",
      "adminUsername": "[parameters('adminUsername')]",
      "linuxConfiguration": {
        "disablePasswordAuthentication": true,
        "ssh": {
          "publicKeys": [
            {
              "path": "[concat('/home/', parameters('adminUsername'), '/.ssh/authorized_keys')]",
              "keyData": "[parameters('adminPublicKey')]"
            }
          ]
        }
      }
    },
    "storageProfile": {
      "imageReference": {
        "publisher": "Canonical",
        "offer": "UbuntuServer",
        "sku": "18.04-LTS",
        "version": "latest"
      },
      "osDisk": {
        "createOption": "fromImage"
      }
    },
    "networkProfile": {
      "networkInterfaces": [
        {
          "id": "[resourceId('Microsoft.Network/networkInterfaces', variables('networkInterfaceName'))]"
        }
      ]
    }
  }
}
]
}
}

```

To run the CLI script, Select **Try** it to open the Azure Cloud shell. To paste the script, right-click the shell, and then select **Paste**:


```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the location (i.e. centralus):" &&
read location &&
echo "Enter the project name (used for generating resource names):" &&
read projectName &&
echo "Enter the administrator username:" &&
read username &&
echo "Enter the SSH public key:" &&
read key &&
az group create --name $resourceGroupName --location "$location" &&
az deployment group create --resource-group $resourceGroupName --template-uri
https://raw.githubusercontent.com/azure/azure-quickstart-templates/master/quickstarts/microsoft.compute/vm-
sshkey/azuredeploy.json --parameters projectName=$projectName adminUsername=$username adminPublicKey="$key"
&&
az vm show --resource-group $resourceGroupName --name "$projectName-vm" --show-details --query publicIps --
output tsv
```

The last Azure CLI command shows the public IP address of the newly created VM. You need the public IP address to connect to the virtual machine. See the next section of this article.

In the previous example, you specified a template stored in GitHub. You can also download or create a template and specify the local path with the `--template-file` parameter.

Here are some additional resources:

- To learn how to develop Resource Manager templates, see [Azure Resource Manager documentation](#).
- To see the Azure virtual machine schemas, see [Azure template reference](#).
- To see more virtual machine template samples, see [Azure Quickstart templates](#).

Connect to virtual machine

You can then SSH to your VM as normal. Provide your own public IP address from the preceding command:

```
ssh <adminUsername>@<ipAddress>
```

Next steps

In this example, you created a basic Linux VM. For more Resource Manager templates that include application frameworks or create more complex environments, browse the [Azure Quickstart templates](#).

To learn more about creating templates, view the JSON syntax and properties for the resources types you deployed:

- [Microsoft.Network/networkSecurityGroups](#)
- [Microsoft.Network/publicIPAddresses](#)
- [Microsoft.Network/virtualNetworks](#)
- [Microsoft.Network/networkInterfaces](#)
- [Microsoft.Compute/virtualMachines](#)

Create a Windows virtual machine from a Resource Manager template

5/28/2021 • 5 minutes to read • [Edit Online](#)

Learn how to create a Windows virtual machine by using an Azure Resource Manager template and Azure PowerShell from the Azure Cloud shell. The template used in this article deploys a single virtual machine running Windows Server in a new virtual network with a single subnet. For creating a Linux virtual machine, see [How to create a Linux virtual machine with Azure Resource Manager templates](#).

An alternative is to deploy the template from the Azure portal. To open the template in the portal, select the **Deploy to Azure** button.



Create a virtual machine

Creating an Azure virtual machine usually includes two steps:

- Create a resource group. An Azure resource group is a logical container into which Azure resources are deployed and managed. A resource group must be created before a virtual machine.
- Create a virtual machine.

The following example creates a VM from an [Azure Quickstart template](#). Here is a copy of the template:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.4.1.14562",
      "templateHash": "8381960602397537918"
    }
  },
  "parameters": {
    "adminUsername": {
      "type": "string",
      "metadata": {
        "description": "Username for the Virtual Machine."
      }
    },
    "adminPassword": {
      "type": "secureString",
      "minLength": 12,
      "metadata": {
        "description": "Password for the Virtual Machine."
      }
    },
    "dnsLabelPrefix": {
      "type": "string",
      "defaultValue": "[toLower(format('{0}-{1}', parameters('vmName'), uniqueString(resourceGroup().id, parameters('vmName'))))]",
      "metadata": {
        "description": "Unique DNS Name for the Public IP used to access the Virtual Machine."
      }
    },
    "publicIpName": {
```

```

    "type": "string",
    "defaultValue": "myPublicIP",
    "metadata": {
      "description": "Name for the Public IP used to access the Virtual Machine."
    }
  },
  "publicIPAllocationMethod": {
    "type": "string",
    "defaultValue": "Dynamic",
    "allowedValues": [
      "Dynamic",
      "Static"
    ],
    "metadata": {
      "description": "Allocation method for the Public IP used to access the Virtual Machine."
    }
  },
  "publicIpSku": {
    "type": "string",
    "defaultValue": "Basic",
    "allowedValues": [
      "Basic",
      "Standard"
    ],
    "metadata": {
      "description": "SKU for the Public IP used to access the Virtual Machine."
    }
  },
  "OSVersion": {
    "type": "string",
    "defaultValue": "2019-Datacenter",
    "allowedValues": [
      "2008-R2-SP1",
      "2012-Datacenter",
      "2012-R2-Datacenter",
      "2016-Nano-Server",
      "2016-Datacenter-with-Containers",
      "2016-Datacenter",
      "2019-Datacenter",
      "2019-Datacenter-Core",
      "2019-Datacenter-Core-smalldisk",
      "2019-Datacenter-Core-with-Containers",
      "2019-Datacenter-Core-with-Containers-smalldisk",
      "2019-Datacenter-smalldisk",
      "2019-Datacenter-with-Containers",
      "2019-Datacenter-with-Containers-smalldisk"
    ],
    "metadata": {
      "description": "The Windows version for the VM. This will pick a fully patched image of this given
Windows version."
    }
  },
  "vmSize": {
    "type": "string",
    "defaultValue": "Standard_D2_v3",
    "metadata": {
      "description": "Size of the virtual machine."
    }
  },
  "location": {
    "type": "string",
    "defaultValue": "[resourceGroup().location]",
    "metadata": {
      "description": "Location for all resources."
    }
  },
  "vmName": {
    "type": "string",
    "defaultValue": "simple-vm"
  }
}

```

```

        defaultvalue : simple-vm ,
        "metadata": {
            "description": "Name of the virtual machine."
        }
    },
},
"functions": [],
"variables": {
    "storageAccountName": "[format('bootdiags{0}', uniqueString(resourceGroup().id))]",
    "nicName": "myVMNic",
    "addressPrefix": "10.0.0.0/16",
    "subnetName": "Subnet",
    "subnetPrefix": "10.0.0.0/24",
    "virtualNetworkName": "MyVNET",
    "networkSecurityGroupName": "default-NSG"
},
"resources": [
    {
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2021-04-01",
        "name": "[variables('storageAccountName')]",
        "location": "[parameters('location')]",
        "sku": {
            "name": "Standard_LRS"
        },
        "kind": "Storage"
    },
    {
        "type": "Microsoft.Network/publicIPAddresses",
        "apiVersion": "2021-02-01",
        "name": "[parameters('publicIpName')]",
        "location": "[parameters('location')]",
        "sku": {
            "name": "[parameters('publicIpSku')]"
        },
        "properties": {
            "publicIPAllocationMethod": "[parameters('publicIPAllocationMethod')]",
            "dnsSettings": {
                "domainNameLabel": "[parameters('dnsLabelPrefix')]"
            }
        }
    },
    {
        "type": "Microsoft.Network/networkSecurityGroups",
        "apiVersion": "2021-02-01",
        "name": "[variables('networkSecurityGroupName')]",
        "location": "[parameters('location')]",
        "properties": {
            "securityRules": [
                {
                    "name": "default-allow-3389",
                    "properties": {
                        "priority": 1000,
                        "access": "Allow",
                        "direction": "Inbound",
                        "destinationPortRange": "3389",
                        "protocol": "Tcp",
                        "sourcePortRange": "*",
                        "sourceAddressPrefix": "*",
                        "destinationAddressPrefix": "*"
                    }
                }
            ]
        }
    },
    {
        "type": "Microsoft.Network/virtualNetworks",
        "apiVersion": "2021-02-01",
        "name": "[variables('virtualNetworkName')]",

```

```

"location": "[parameters('location')]",
"properties": {
  "addressSpace": {
    "addressPrefixes": [
      "[variables('addressPrefix')]"
    ]
  }
}
},
{
  "type": "Microsoft.Network/virtualNetworks/subnets",
  "apiVersion": "2021-02-01",
  "name": "[format('{0}/{1}', variables('virtualNetworkName'), variables('subnetName'))]",
  "properties": {
    "addressPrefix": "[variables('subnetPrefix')]",
    "networkSecurityGroup": {
      "id": "[resourceId('Microsoft.Network/networkSecurityGroups',
variables('networkSecurityGroupName'))]"
    }
  },
  "dependsOn": [
    "[resourceId('Microsoft.Network/networkSecurityGroups', variables('networkSecurityGroupName'))]",
    "[resourceId('Microsoft.Network/virtualNetworks', variables('virtualNetworkName'))]"
  ]
},
{
  "type": "Microsoft.Network/networkInterfaces",
  "apiVersion": "2021-02-01",
  "name": "[variables('nicName')]",
  "location": "[parameters('location')]",
  "properties": {
    "ipConfigurations": [
      {
        "name": "ipconfig1",
        "properties": {
          "privateIPAllocationMethod": "Dynamic",
          "publicIPAddress": {
            "id": "[resourceId('Microsoft.Network/publicIPAddresses', parameters('publicIpName'))]"
          },
          "subnet": {
            "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets',
variables('virtualNetworkName'), variables('subnetName'))]"
          }
        }
      }
    ]
  },
  "dependsOn": [
    "[resourceId('Microsoft.Network/publicIPAddresses', parameters('publicIpName'))]",
    "[resourceId('Microsoft.Network/virtualNetworks/subnets', variables('virtualNetworkName'),
variables('subnetName'))]"
  ]
},
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2021-03-01",
  "name": "[parameters('vmName')]",
  "location": "[parameters('location')]",
  "properties": {
    "hardwareProfile": {
      "vmSize": "[parameters('vmSize')]"
    },
    "osProfile": {
      "computerName": "[parameters('vmName')]",
      "adminUsername": "[parameters('adminUsername')]",
      "adminPassword": "[parameters('adminPassword')]"
    },
    "storageProfile": {
      "imageReference": {

```

```

    "publisher": "MicrosoftWindowsServer",
    "offer": "WindowsServer",
    "sku": "[parameters('OSVersion')]",
    "version": "latest"
  },
  "osDisk": {
    "createOption": "FromImage",
    "managedDisk": {
      "storageAccountType": "StandardSSD_LRS"
    }
  },
  "dataDisks": [
    {
      "diskSizeGB": 1023,
      "lun": 0,
      "createOption": "Empty"
    }
  ]
},
"networkProfile": {
  "networkInterfaces": [
    {
      "id": "[resourceId('Microsoft.Network/networkInterfaces', variables('nicName'))]"
    }
  ]
},
"diagnosticsProfile": {
  "bootDiagnostics": {
    "enabled": true,
    "storageUri": "[reference(resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName'))).primaryEndpoints.blob]"
  }
}
},
"dependsOn": [
  "[resourceId('Microsoft.Network/networkInterfaces', variables('nicName'))]",
  "[resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName'))]"
]
}
],
"outputs": {
  "hostname": {
    "type": "string",
    "value": "[reference(resourceId('Microsoft.Network/publicIPAddresses',
parameters('publicIpName'))).dnsSettings.fqdn]"
  }
}
}
}

```

To run the PowerShell script, Select **Try** it to open the Azure Cloud shell. To paste the script, right-click the shell, and then select **Paste**:

```

$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
$location = Read-Host -Prompt "Enter the location (i.e. centralus)"
$adminUsername = Read-Host -Prompt "Enter the administrator username"
$adminPassword = Read-Host -Prompt "Enter the administrator password" -AsSecureString
$dnsLabelPrefix = Read-Host -Prompt "Enter an unique DNS name for the public IP"

New-AzResourceGroup -Name $resourceGroupName -Location "$location"
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -TemplateUri "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.compute/vm-simple-windows/azuredeploy.json" `
  -adminUsername $adminUsername `
  -adminPassword $adminPassword `
  -dnsLabelPrefix $dnsLabelPrefix

(Get-AzVm -ResourceGroupName $resourceGroupName).name

```

If you choose to install and use the PowerShell locally instead of from the Azure Cloud shell, this tutorial requires the Azure PowerShell module. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see [Install Azure PowerShell module](#). If you're running PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

In the previous example, you specified a template stored in GitHub. You can also download or create a template and specify the local path with the `--template-file` parameter.

Here are some additional resources:

- To learn how to develop Resource Manager templates, see [Azure Resource Manager documentation](#).
- To see the Azure virtual machine schemas, see [Azure template reference](#).
- To see more virtual machine template samples, see [Azure Quickstart templates](#).

Connect to the virtual machine

The last PowerShell command from the previous script shows the virtual machine name. To connect to the virtual machine, see [How to connect and sign on to an Azure virtual machine running Windows](#).

Next Steps

- If there were issues with the deployment, you might take a look at [Troubleshoot common Azure deployment errors with Azure Resource Manager](#).
- Learn how to create and manage a virtual machine in [Create and manage Windows VMs with the Azure PowerShell module](#).

To learn more about creating templates, view the JSON syntax and properties for the resources types you deployed:

- [Microsoft.Network/publicIPAddresses](#)
- [Microsoft.Network/virtualNetworks](#)
- [Microsoft.Network/networkInterfaces](#)
- [Microsoft.Compute/virtualMachines](#)

Azure Functions developer guide

6/8/2021 • 11 minutes to read • [Edit Online](#)

In Azure Functions, specific functions share a few core technical concepts and components, regardless of the language or binding you use. Before you jump into learning details specific to a given language or binding, be sure to read through this overview that applies to all of them.

This article assumes that you've already read the [Azure Functions overview](#).

Function code

A *function* is the primary concept in Azure Functions. A function contains two important pieces - your code, which can be written in a variety of languages, and some config, the function.json file. For compiled languages, this config file is generated automatically from annotations in your code. For scripting languages, you must provide the config file yourself.

The function.json file defines the function's trigger, bindings, and other configuration settings. Every function has one and only one trigger. The runtime uses this config file to determine the events to monitor and how to pass data into and return data from a function execution. The following is an example function.json file.

```
{
  "disabled":false,
  "bindings":[
    // ... bindings here
    {
      "type": "bindingType",
      "direction": "in",
      "name": "myParamName",
      // ... more depending on binding
    }
  ]
}
```

For more information, see [Azure Functions triggers and bindings concepts](#).

The `bindings` property is where you configure both triggers and bindings. Each binding shares a few common settings and some settings which are specific to a particular type of binding. Every binding requires the following settings:

PROPERTY	VALUES	TYPE	COMMENTS
type	Name of binding. For example, <code>queueTrigger</code> .	string	
direction	<code>in</code> , <code>out</code>	string	Indicates whether the binding is for receiving data into the function or sending data from the function.

PROPERTY	VALUES	TYPE	COMMENTS
name	Function identifier. For example, <code>myQueue</code> .	string	The name that is used for the bound data in the function. For C#, this is an argument name; for JavaScript, it's the key in a key/value list.

Function app

A function app provides an execution context in Azure in which your functions run. As such, it is the unit of deployment and management for your functions. A function app is comprised of one or more individual functions that are managed, deployed, and scaled together. All of the functions in a function app share the same pricing plan, deployment method, and runtime version. Think of a function app as a way to organize and collectively manage your functions. To learn more, see [How to manage a function app](#).

NOTE

All functions in a function app must be authored in the same language. In [previous versions](#) of the Azure Functions runtime, this wasn't required.

Folder structure

The code for all the functions in a specific function app is located in a root project folder that contains a host configuration file. The `host.json` file contains runtime-specific configurations and is in the root folder of the function app. A `bin` folder contains packages and other library files that the function app requires. Specific folder structures required by the function app depend on language:

- [C# compiled \(.csproj\)](#)
- [C# script \(.csx\)](#)
- [F# script](#)
- [Java](#)
- [JavaScript](#)
- [Python](#)

In version 2.x and higher of the Functions runtime, all functions in the function app must share the same language stack.

The above is the default (and recommended) folder structure for a Function app. If you wish to change the file location of a function's code, modify the `scriptFile` section of the `function.json` file. We also recommend using [package deployment](#) to deploy your project to your function app in Azure. You can also use existing tools like [continuous integration and deployment](#) and Azure DevOps.

NOTE

If deploying a package manually, make sure to deploy your `host.json` file and function folders directly to the `wwwroot` folder. Do not include the `wwwroot` folder in your deployments. Otherwise, you end up with `wwwroot\wwwroot` folders.

Use local tools and publishing

Function apps can be authored and published using a variety of tools, including [Visual Studio](#), [Visual Studio Code](#), [IntelliJ](#), [Eclipse](#), and the [Azure Functions Core Tools](#). For more information, see [Code and test Azure Functions locally](#).

How to edit functions in the Azure portal

The Functions editor built into the Azure portal lets you update your code and your *function.json* file directly inline. This is recommended only for small changes or proofs of concept - best practice is to use a local development tool like VS Code.

Parallel execution

When multiple triggering events occur faster than a single-threaded function runtime can process them, the runtime may invoke the function multiple times in parallel. If a function app is using the [Consumption hosting plan](#), the function app could scale out automatically. Each instance of the function app, whether the app runs on the Consumption hosting plan or a regular [App Service hosting plan](#), might process concurrent function invocations in parallel using multiple threads. The maximum number of concurrent function invocations in each function app instance varies based on the type of trigger being used as well as the resources used by other functions within the function app.

Functions runtime versioning

You can configure the version of the Functions runtime using the `FUNCTIONS_EXTENSION_VERSION` app setting. For example, the value "~3" indicates that your function app will use 3.x as its major version. Function apps are upgraded to each new minor version as they are released. For more information, including how to view the exact version of your function app, see [How to target Azure Functions runtime versions](#).

Repositories

The code for Azure Functions is open source and stored in GitHub repositories:

- [Azure Functions](#)
- [Azure Functions host](#)
- [Azure Functions portal](#)
- [Azure Functions templates](#)
- [Azure WebJobs SDK](#)
- [Azure WebJobs SDK Extensions](#)

Bindings

Here is a table of all supported bindings.

This table shows the bindings that are supported in the major versions of the Azure Functions runtime:

TYPE	1.X	2.X AND HIGHER ¹	TRIGGER	INPUT	OUTPUT
Blob storage	✓	✓	✓	✓	✓
Azure Cosmos DB	✓	✓	✓	✓	✓
Dapr³		✓	✓	✓	✓
Event Grid	✓	✓	✓		✓
Event Hubs	✓	✓	✓		✓

TYPE	1.X	2.X AND HIGHER	TRIGGER	INPUT	OUTPUT
HTTP & webhooks	✓	✓	✓		✓
IoT Hub	✓	✓	✓		✓
Kafka ²		✓	✓		✓
Mobile Apps	✓			✓	✓
Notification Hubs	✓				✓
Queue storage	✓	✓	✓		✓
RabbitMQ ²		✓	✓		✓
SendGrid	✓	✓			✓
Service Bus	✓	✓	✓		✓
SignalR		✓		✓	✓
Table storage	✓	✓		✓	✓
Timer	✓	✓	✓		
Twilio	✓	✓			✓

¹ Starting with the version 2.x runtime, all bindings except HTTP and Timer must be registered. See [Register binding extensions](#).

² Triggers aren't supported in the Consumption plan. Requires [runtime-driven triggers](#).

³ Supported only in Kubernetes, IoT Edge, and other self-hosted modes only.

Having issues with errors coming from the bindings? Review the [Azure Functions Binding Error Codes](#) documentation.

Connections

Your function project references connection information by name from its configuration provider. It does not directly accept the connection details, allowing them to be changed across environments. For example, a trigger definition might include a `connection` property. This might refer to a connection string, but you cannot set the connection string directly in a `function.json`. Instead, you would set `connection` to the name of an environment variable that contains the connection string.

The default configuration provider uses environment variables. These might be set by [Application Settings](#) when running in the Azure Functions service, or from the [local settings file](#) when developing locally.

Connection values

When the connection name resolves to a single exact value, the runtime identifies the value as a *connection string*, which typically includes a secret. The details of a connection string are defined by the service to which

you wish to connect.

However, a connection name can also refer to a collection of multiple configuration items. Environment variables can be treated as a collection by using a shared prefix that ends in double underscores `__`. The group can then be referenced by setting the connection name to this prefix.

For example, the `connection` property for a Azure Blob trigger definition might be `Storage1`. As long as there is no single string value configured with `Storage1` as its name, `Storage1__serviceUri` would be used for the `serviceUri` property of the connection. The connection properties are different for each service. Refer to the documentation for the extension that uses the connection.

Configure an identity-based connection

Some connections in Azure Functions are configured to use an identity instead of a secret. Support depends on the extension using the connection. In some cases, a connection string may still be required in Functions even though the service to which you are connecting supports identity-based connections.

Identity-based connections are supported by the following trigger and binding extensions in all plans:

NOTE

Identity-based connections are not supported with Durable Functions.

EXTENSION NAME	EXTENSION VERSION
Azure Blob	Version 5.0.0-beta1 or later
Azure Queue	Version 5.0.0-beta1 or later
Azure Event Hubs	Version 5.0.0-beta1 or later
Azure Service Bus	Version 5.0.0-beta2 or later

The storage connections used by the Functions runtime (`AzureWebJobsStorage`) may also be configured using an identity-based connection. See [Connecting to host storage with an identity](#) below.

When hosted in the Azure Functions service, identity-based connections use a [managed identity](#). The system-assigned identity is used by default. When run in other contexts, such as local development, your developer identity is used instead, although this can be customized using alternative connection parameters.

Grant permission to the identity

Whatever identity is being used must have permissions to perform the intended actions. This is typically done by assigning a role in Azure RBAC or specifying the identity in an access policy, depending on the service to which you are connecting. Refer to the documentation for each extension on what permissions are needed and how they can be set.

IMPORTANT

Some permissions might be exposed by the target service that are not necessary for all contexts. Where possible, adhere to the [principle of least privilege](#), granting the identity only required privileges. For example, if the app just needs to read from a blob, use the [Storage Blob Data Reader](#) role as the [Storage Blob Data Owner](#) includes excessive permissions for a read operation. The following roles cover the primary permissions needed for each extension in normal operation:

SERVICE	EXAMPLE BUILT-IN ROLES
Azure Blobs	Storage Blob Data Reader , Storage Blob Data Owner
Azure Queues	Storage Queue Data Reader , Storage Queue Data Message Processor , Storage Queue Data Message Sender , Storage Queue Data Contributor
Event Hubs	Azure Event Hubs Data Receiver , Azure Event Hubs Data Sender , Azure Event Hubs Data Owner
Service Bus	Azure Service Bus Data Receiver , Azure Service Bus Data Sender , Azure Service Bus Data Owner

Connection properties

An identity-based connection for an Azure service accepts the following properties:

PROPERTY	REQUIRED FOR EXTENSIONS	ENVIRONMENT VARIABLE	DESCRIPTION
Service URI	Azure Blob ¹ , Azure Queue	<code><CONNECTION_NAME_PREFIX>_serviceUri</code>	The data plane URI of the service to which you are connecting.
Fully Qualified Namespace	Event Hubs, Service Bus	<code><CONNECTION_NAME_PREFIX>_fullyQualifiedNamespace</code>	The fully qualified Event Hubs and Service Bus namespace.

¹ Both blob and queue service URI's are required for Azure Blob.

Additional options may be supported for a given connection type. Please refer to the documentation for the component making the connection.

Local development

When running locally, the above configuration tells the runtime to use your local developer identity. The connection will attempt to get a token from the following locations, in order:

- A local cache shared between Microsoft applications
- The current user context in Visual Studio
- The current user context in Visual Studio Code
- The current user context in the Azure CLI

If none of these options are successful, an error will occur.

In some cases, you may wish to specify use of a different identity. You can add configuration properties for the connection that point to the alternate identity.

NOTE

The following configuration options are not supported when hosted in the Azure Functions service.

To connect using an Azure Active Directory service principal with a client ID and secret, define the connection with the following required properties in addition to the [Connection properties](#) above:

PROPERTY	ENVIRONMENT VARIABLE	DESCRIPTION
Tenant ID	<CONNECTION_NAME_PREFIX>__tenantId	The Azure Active Directory tenant (directory) ID.
Client ID	<CONNECTION_NAME_PREFIX>__clientId	The client (application) ID of an app registration in the tenant.
Client secret	<CONNECTION_NAME_PREFIX>__clientSecret	A client secret that was generated for the app registration.

Example of `local.settings.json` properties required for identity-based connection with Azure Blob:

```
{
  "IsEncrypted": false,
  "Values": {
    "<CONNECTION_NAME_PREFIX>__serviceUri": "<serviceUri>",
    "<CONNECTION_NAME_PREFIX>__tenantId": "<tenantId>",
    "<CONNECTION_NAME_PREFIX>__clientId": "<clientId>",
    "<CONNECTION_NAME_PREFIX>__clientSecret": "<clientSecret>"
  }
}
```

Connecting to host storage with an identity

Azure Functions by default uses the `AzureWebJobsStorage` connection for core behaviors such as coordinating singleton execution of timer triggers and default app key storage. This can be configured to leverage an identity as well.

Caution

Some apps reuse `AzureWebJobsStorage` for storage connections in their triggers, bindings, and/or function code. Make sure that all uses of `AzureWebJobsStorage` are able to use the identity-based connection format before changing this connection from a connection string.

To configure the connection in this way, make sure the app's identity has the [Storage Blob Data Owner](#) role in order to support the core host functionality. You may need additional permissions if you use "AzureWebJobsStorage" for any other purposes.

If using a storage account that uses the default DNS suffix and service name for global Azure, following the

`https://<accountName>.blob/queue/file/table.core.windows.net` format, you can set

`AzureWebJobsStorage__accountName` to the name of your storage account.

If instead using a storage account in a sovereign cloud or with custom DNS, set

`AzureWebJobsStorage__serviceUri` to the URI for your blob service. If "AzureWebJobsStorage" will be used for any other service, you may instead specify

`AzureWebJobsStorage__blobServiceUri`,

`AzureWebJobsStorage__queueServiceUri`, and `AzureWebJobsStorage__tableServiceUri` separately.

Reporting Issues

ITEM	DESCRIPTION	LINK
Runtime	Script Host, Triggers & Bindings, Language Support	File an Issue
Templates	Code Issues with Creation Template	File an Issue

ITEM	DESCRIPTION	LINK
Portal	User Interface or Experience Issue	File an Issue

Next steps

For more information, see the following resources:

- [Azure Functions triggers and bindings](#)
- [Code and test Azure Functions locally](#)
- [Best Practices for Azure Functions](#)
- [Azure Functions C# developer reference](#)
- [Azure Functions Node.js developer reference](#)

Create a Service Fabric cluster in Azure using the Azure portal

11/2/2020 • 11 minutes to read • [Edit Online](#)

This is a step-by-step guide that walks you through the steps of setting up a Service Fabric cluster (Linux or Windows) in Azure using the Azure portal. This guide walks you through the following steps:

- Create a cluster in Azure through the Azure portal.
- Authenticate administrators using certificates.

NOTE

For more advanced security options, such as user authentication with Azure Active Directory and setting up certificates for application security, [create your cluster using Azure Resource Manager](#).

Cluster security

Certificates are used in Service Fabric to provide authentication and encryption to secure various aspects of a cluster and its applications. For more information on how certificates are used in Service Fabric, see [Service Fabric cluster security scenarios](#).

If this is the first time you are creating a service fabric cluster or are deploying a cluster for test workloads, you can skip to the next section (**Create cluster in the Azure portal**) and have the system generate certificates needed for your clusters that run test workloads. If you are setting up a cluster for production workloads, then continue reading.

Cluster and server certificate (required)

This certificate is required to secure a cluster and prevent unauthorized access to it. It provides cluster security in a couple ways:

- **Cluster authentication:** Authenticates node-to-node communication for cluster federation. Only nodes that can prove their identity with this certificate can join the cluster.
- **Server authentication:** Authenticates the cluster management endpoints to a management client, so that the management client knows it is talking to the real cluster. This certificate also provides TLS for the HTTPS management API and for Service Fabric Explorer over HTTPS.

To serve these purposes, the certificate must meet the following requirements:

- The certificate must contain a private key.
- The certificate must be created for key exchange, exportable to a Personal Information Exchange (.pfx) file.
- The certificate's **subject name must match the domain** used to access the Service Fabric cluster. This is required to provide TLS for the cluster's HTTPS management endpoints and Service Fabric Explorer. You cannot obtain a TLS/SSL certificate from a certificate authority (CA) for the `.cloudapp.azure.com` domain. Acquire a custom domain name for your cluster. When you request a certificate from a CA the certificate's subject name must match the custom domain name used for your cluster.

Client authentication certificates

Additional client certificates authenticate administrators for cluster management tasks. Service Fabric has two access levels: **admin** and **read-only user**. At minimum, a single certificate for administrative access should be used. For additional user-level access, a separate certificate must be provided. For more information on access

roles, see [role-based access control for Service Fabric clients](#).

You do not need to upload Client authentication certificates to Key Vault to work with Service Fabric. These certificates only need to be provided to users who are authorized for cluster management.

NOTE

Azure Active Directory is the recommended way to authenticate clients for cluster management operations. To use Azure Active Directory, you must [create a cluster using Azure Resource Manager](#).

Application certificates (optional)

Any number of additional certificates can be installed on a cluster for application security purposes. Before creating your cluster, consider the application security scenarios that require a certificate to be installed on the nodes, such as:

- Encryption and decryption of application configuration values
- Encryption of data across nodes during replication

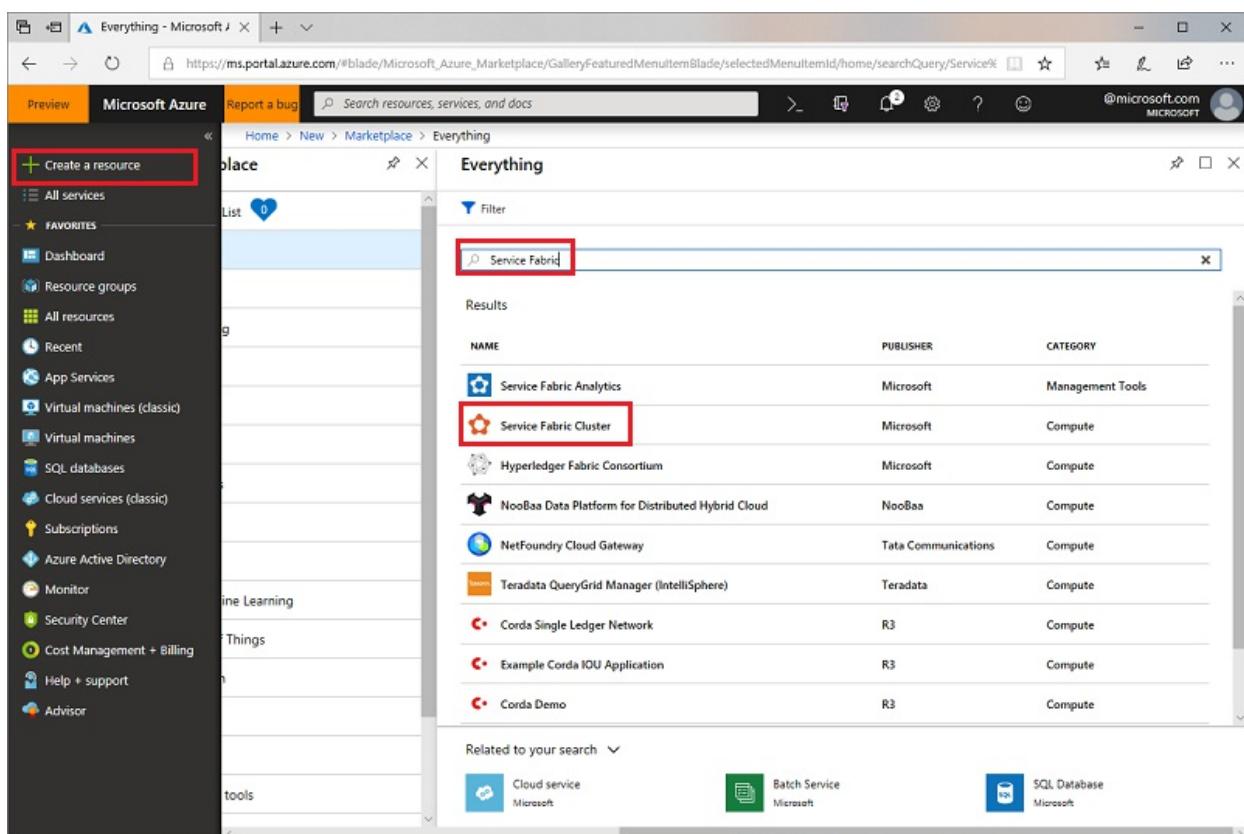
Application certificates cannot be configured when [creating a cluster through the Azure portal](#). To configure application certificates at cluster setup time, you must [create a cluster using Azure Resource Manager](#). You can also add application certificates to the cluster after it has been created.

Create cluster in the Azure portal

Creating a production cluster to meet your application needs involves some planning, to help you with that, it is strongly recommended that you read and understand the [Service Fabric Cluster planning considerations](#) document.

Search for the Service Fabric cluster resource

Sign in to the [Azure portal](#). Click **Create a resource** to add a new resource template. Search for the Service Fabric Cluster template in the **Marketplace** under **Everything**. Select **Service Fabric Cluster** from the list.



Navigate to the Service Fabric Cluster blade, and click **Create**.

The Create Service Fabric cluster blade has the following four steps:

1. Basics

The screenshot shows the 'Create Service Fabric cluster' blade with the 'Basics' tab selected. The left sidebar lists four steps: 1. Basics (Done), 2. Cluster configuration, 3. Security, and 4. Summary. The main area contains the following fields:

- Cluster name:** sfdocscluster (with a green checkmark)
- Operating system:** WindowsServer 2016-Datacenter-with-Cr (with a dropdown arrow)
- Default VM credentials:**
 - User name:** sfuser (with a green checkmark)
 - Password:** masked with dots
 - Confirm password:** masked with dots
- Subscription:** Windows Azure Internal Consumption (0) (with a dropdown arrow)
- Resource group:** Create new (selected) / Use existing (unselected). sfdocscluster (with a green checkmark)
- Location:** East US (with a dropdown arrow)

An **OK** button is located at the bottom of the form.

In the Basics blade, you need to provide the basic details for your cluster.

1. Enter the name of your cluster.
2. Enter a **User name** and **Password** for Remote Desktop for the VMs.
3. Make sure to select the **Subscription** that you want your cluster to be deployed to, especially if you have multiple subscriptions.
4. Create a new **Resource group**. It is best to give it the same name as the cluster, since it helps in finding them later, especially when you are trying to make changes to your deployment or delete your cluster.

NOTE

Although you can decide to use an existing resource group, it is a good practice to create a new resource group. This makes it easy to delete clusters and all the resources it uses.

5. Select the **Location** in which you want to create the cluster. If you are planning to use an existing certificate that you have already uploaded to a key vault, You must use the same region that your Key vault is in.

2. Cluster configuration

The screenshot shows the 'Create Service Fabric cluster' wizard with three panels:

- 1 Basics Done** (marked as complete with a green checkmark)
- 2 Cluster configuration** (Set up cluster configuration) - This is the active panel, showing:
 - Node types**: Node types define the scale sets that will be used to manage your cluster. You specify between 1 and 3 in the portal. [Learn more about node types](#)
 - * Node type count**: Radio buttons for 1 (selected), 2, and 3.
 - * Node type 1 (Primary)**: [Configure required settings](#)
 - [+ Show optional settings](#)
- 3 Security** (Configure security settings)
- 4 Summary** (Review, view template, create)

The 'Node type configuration' panel for 'Node type 1 (Primary)' includes the following settings:

- * Node type name**:
- Durability tier**:
- * Virtual machine size**:
- Single node cluster**
- Initial VM scale set capacity**:
- Custom endpoints**:
- Enable reverse proxy**
- Reverse proxy port**:
- Configure advanced settings**

Configure your cluster nodes. Node types define the VM sizes, the number of VMs, and their properties. Your cluster can have more than one node type, but the primary node type (the first one that you define on the portal) must have at least five VMs, as this is the node type where Service Fabric system services are placed. Do not configure **Placement Properties** because a default placement property of "NodeTypeName" is added automatically.

NOTE

A common scenario for multiple node types is an application that contains a front-end service and a back-end service. You want to put the front-end service on smaller VMs (VM sizes like D2_V2) with ports open to the Internet, and put the back-end service on larger VMs (with VM sizes like D3_V2, D6_V2, D15_V2, and so on) with no Internet-facing ports open.

1. Choose a name for your node type (1 to 12 characters containing only letters and numbers).
2. The minimum **size** of VMs for the primary node type is driven by the **Durability tier** you choose for the cluster. The default for the durability tier is bronze. For more information on durability, see [how to choose the Service Fabric cluster durability](#).
3. Select the **Virtual machine size**. D-series VMs have SSD drives and are highly recommended for stateful applications. Do not use any VM SKU that has partial cores or have less than 10 GB of available disk capacity. Refer to [service fabric cluster planning consideration document](#) for help in selecting the VM size.
4. **Single node cluster** and **three node clusters** are meant for test use only. They are not supported for any running production workloads.
5. Choose the **Initial virtual machine scale set capacity** for the node type. You can scale in or out the number of VMs in a node type later on, but on the primary node type, the minimum is five for production workloads. Other node types can have a minimum of one VM. The minimum **number** of VMs for the primary node type drives the **reliability** of your cluster.
6. Configure **Custom endpoints**. This field allows you to enter a comma-separated list of ports that you want

to expose through the Azure Load Balancer to the public Internet for your applications. For example, if you plan to deploy a web application to your cluster, enter "80" here to allow traffic on port 80 into your cluster. For more information on endpoints, see [communicating with applications](#)

7. **Enable reverse proxy.** The [Service Fabric reverse proxy](#) helps microservices running in a Service Fabric cluster discover and communicate with other services that have http endpoints.
8. Back in the **Cluster configuration** blade, under **+Show optional settings**, configure cluster **diagnostics**. By default, diagnostics are enabled on your cluster to assist with troubleshooting issues. If you want to disable diagnostics change the **Status** toggle to **Off**. Turning off diagnostics is **not** recommended. If you already have Application Insights project created, then give its key, so that the application traces are routed to it.
9. **Include DNS service.** The [DNS service](#) an optional service that enables you to find other services using the DNS protocol.
10. Select the **Fabric upgrade mode** you want set your cluster to. Select **Automatic**, if you want the system to automatically pick up the latest available version and try to upgrade your cluster to it. Set the mode to **Manual**, if you want to choose a supported version. For more details on the Fabric upgrade mode see the [Service Fabric Cluster Upgrade document](#).

NOTE

We support only clusters that are running supported versions of Service Fabric. By selecting the **Manual** mode, you are taking on the responsibility to upgrade your cluster to a supported version.

3. Security

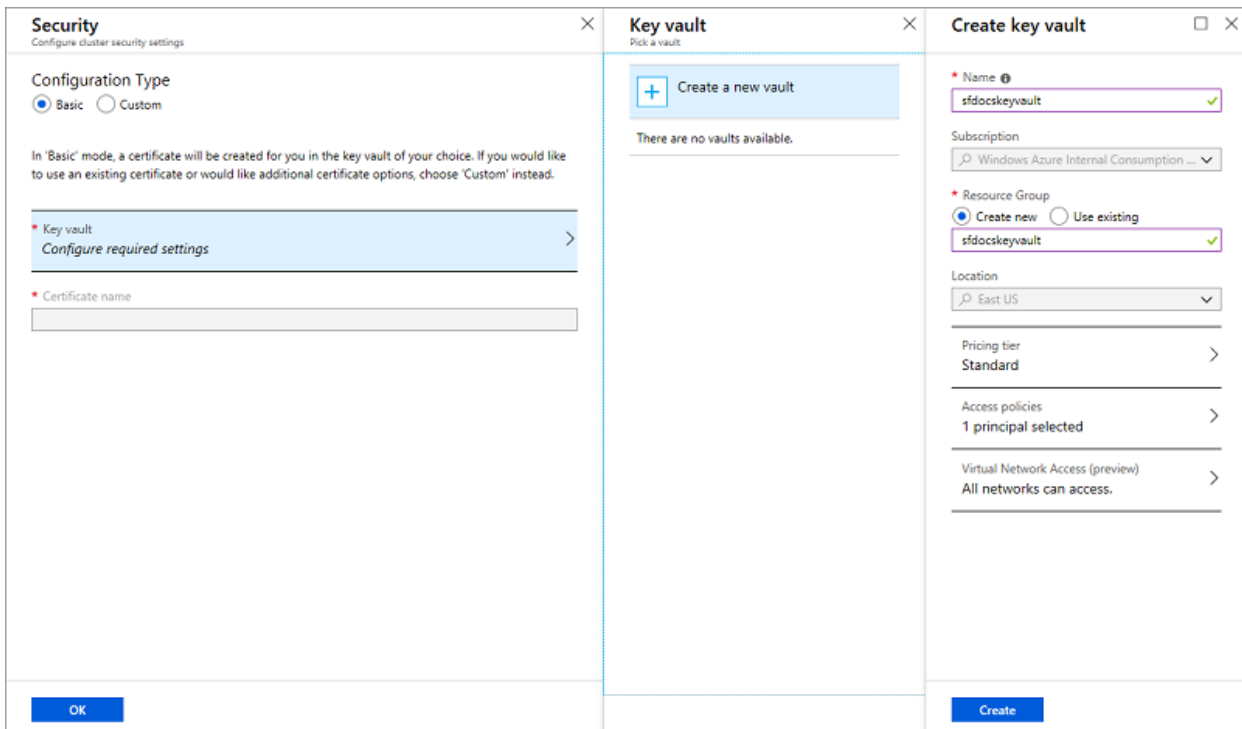
The screenshot shows the 'Create Service Fabric cluster' wizard with the 'Security' step selected. The left sidebar shows four steps: 1. Basics (Done), 2. Cluster configuration (Done), 3. Security (Configure security settings), and 4. Summary (Review, view template, create). The main panel is titled 'Security' and 'Configure cluster security settings'. It has two radio buttons for 'Configuration Type': 'Basic' (selected) and 'Custom'. Below this is a note: 'In "Basic" mode, a certificate will be created for you in the key vault of your choice. If you would like to use an existing certificate or would like additional certificate options, choose "Custom" instead.' There are two input fields: 'Key vault' with a dropdown arrow and 'Certificate name' with a text box. An 'OK' button is at the bottom.

To make setting up a secure test cluster easy for you, we have provided the **Basic** option. If you already have a certificate and have uploaded it to your [key vault](#) (and enabled the key vault for deployment), then use the **Custom** option

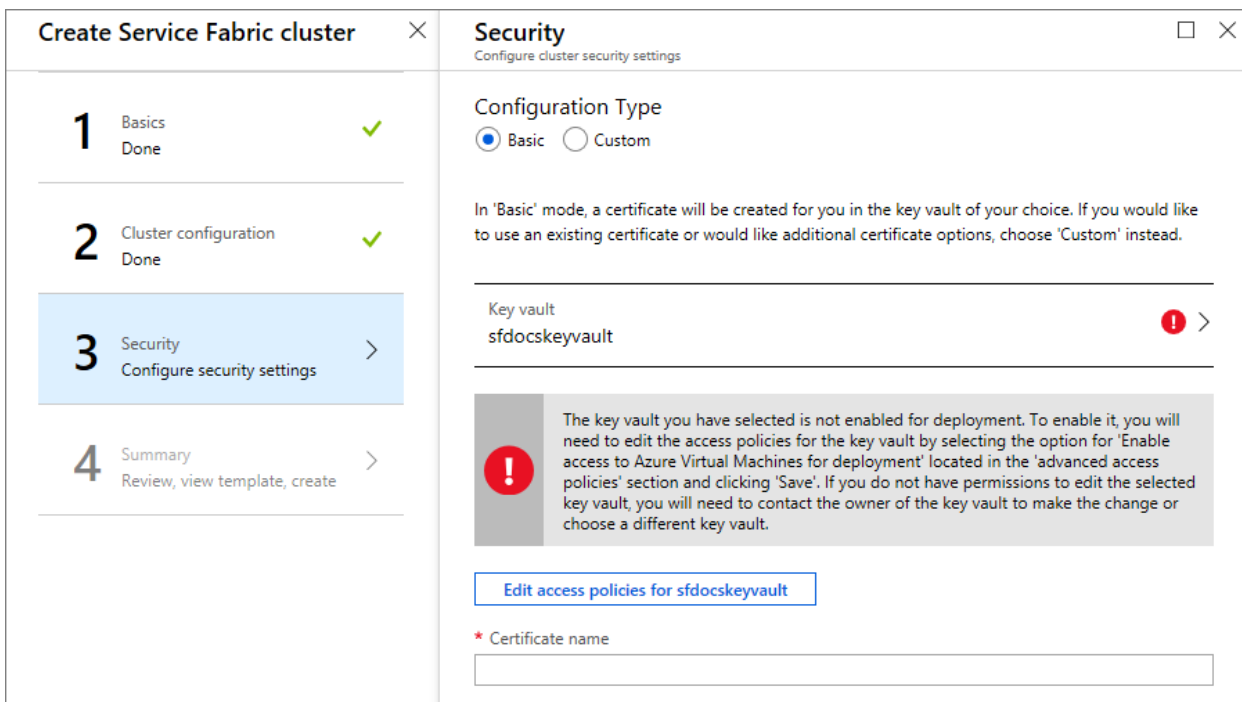
Basic Option

Follow the screens to add or reuse an existing key vault and add a certificate. The addition of the certificate is a synchronous process and so you will have to wait for the certificate to be created.

Resist the temptation of navigating away from the screen until the preceding process is completed.



Now that the key vault is created, edit the access policies for your key vault.



Click on the **Edit access policies**, then **Show advanced access policies** and enable access to Azure Virtual Machines for deployment. It is recommended that you enable the template deployment as well. Once you have made your selections, do not forget to click the **Save** button and close out of the **Access policies** pane.

Create Service Fabric cluster ×

Security
Configure cluster security settings

Configuration Type
 Basic Custom

In 'Basic' mode, a certificate will be created for you in the key vault of your choice. If you would like to use an existing certificate or would like additional certificate options, choose 'Custom' instead.

Key vault
sfdocskeyvault ⓘ

The key vault you have selected is not enabled for deployment. To enable it, you will need to edit the access policies for the key vault by selecting the option for 'Enable access to Azure Virtual Machines for deployment' located in the 'advanced access policies' section and clicking 'Save'. If you do not have permissions to edit the selected key vault, you will need to contact the owner of the key vault to make the change or choose a different key vault.

[Edit access policies for sfdocskeyvault](#)

* Certificate name

Access policies □ ×

Save Discard Refresh

Click to hide advanced access policies

Enable access to Azure Virtual Machines for deployment ⓘ

Enable access to Azure Resource Manager for template deployment ⓘ

Enable access to Azure Disk Encryption for volume encryption ⓘ

+ Add new ...

USER ...

Enter the name of the certificate and click OK.

Create Service Fabric cluster ×

Security
Configure cluster security settings

Configuration Type
 Basic Custom

In 'Basic' mode, a certificate will be created for you in the key vault of your choice. If you would like to use an existing certificate or would like additional certificate options, choose 'Custom' instead.

* Key vault
sfdocskeyvault ⓘ

* Certificate name
sfcluster-eastus ✓

OK

Custom Option

Skip this section, if you have already performed the steps in the **Basic** Option.

Security
□ ×

Configure cluster security settings

Configuration Type

Basic
 Custom

Primary certificate

* Source key vault ⓘ

* Certificate URL ⓘ

* Certificate thumbprint ⓘ

[- Hide advanced settings](#)

Secondary certificate (optional)

SOURCE KEY VAULT	CERTIFICATE URL	CERTIFICATE THUMBPRINT
A secondary certificate has not been added.		
<input style="width: 100%; border: 1px solid #0070c0;" type="button" value="Add a secondary certificate"/>		

Client certificates (optional)

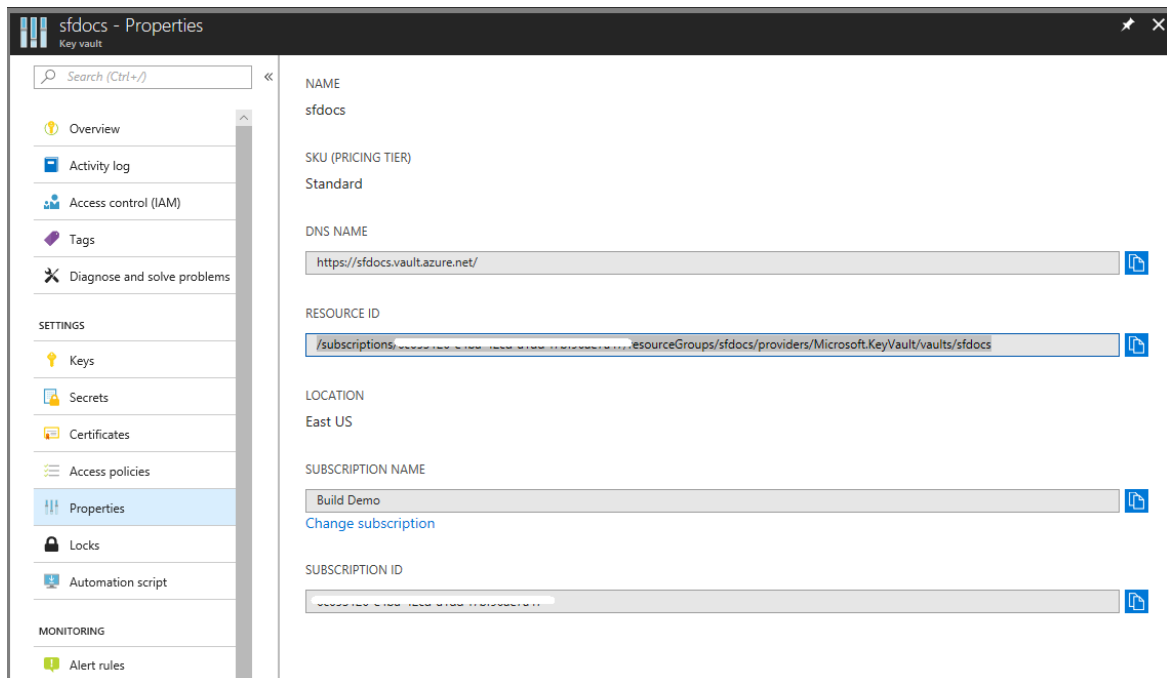
CERTIFICATE TYPE	SUBJECT NAME (OPTIONAL)	THUMBPRINT
No client certificates have been added.		
<input style="width: 100%; border: 1px solid #0070c0;" type="button" value="Add a client certificate"/>		

Azure Active Directory (optional)

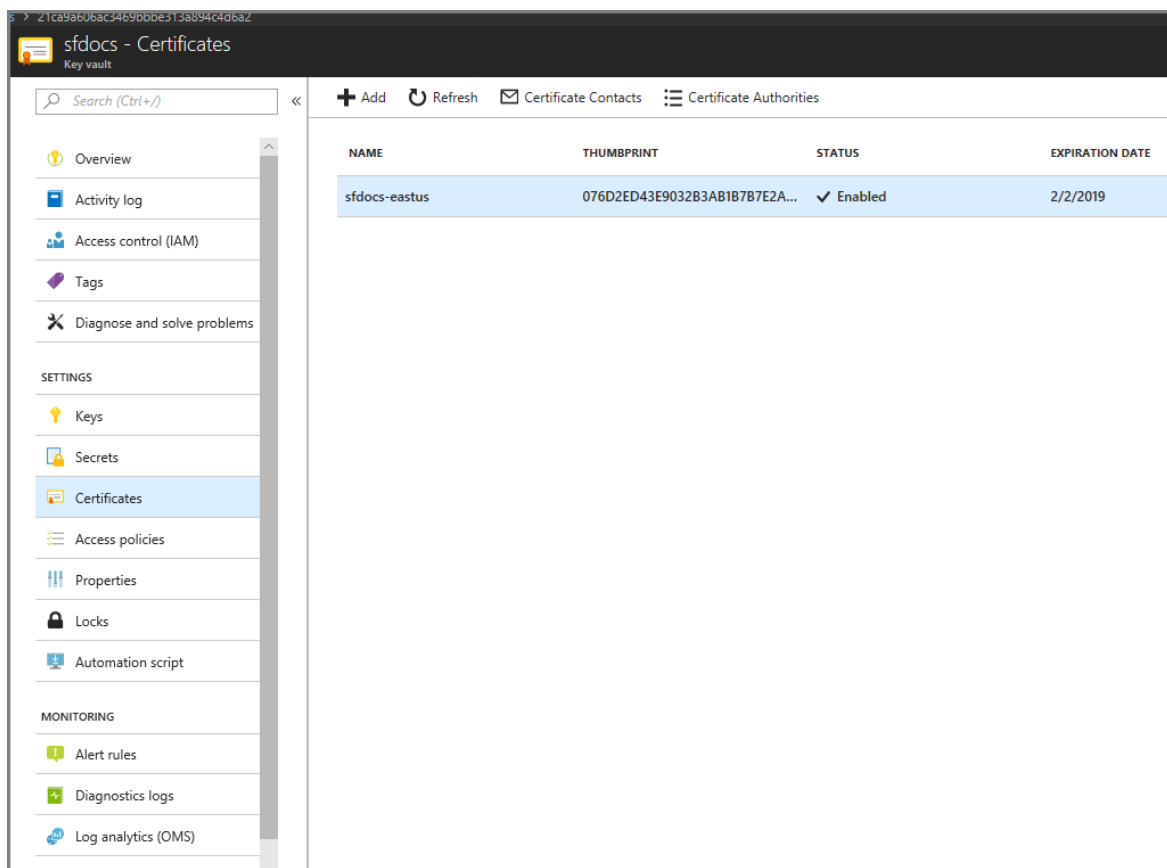
TENANT ID	CLUSTER APPLICATION	CLIENT APPLICATION
An Azure Active Directory has not been added.		
<input style="width: 100%; border: 1px solid #0070c0;" type="button" value="Add an Azure Active Directory"/>		

You need the Source key vault, Certificate URL, and Certificate thumbprint information to complete the security page. If you do not have it handy, open up another browser window and in the Azure portal do the following

1. Navigate to your key vault service.
2. Select the "Properties" tab and copy the 'RESOURCE ID' to "Source key vault" on the other browser window



3. Now, select the "Certificates" tab.
4. Click on certificate thumbprint, which takes you to the Versions page.
5. Click on the GUIDs you see under the current Version.



6. You should now be on the screen like below. Copy the hexadecimal SHA-1 Thumbprint to "Certificate thumbprint" on the other browser window
7. Copy the 'Secret Identifier' to the "Certificate URL" on other browser window.

21ca9a606ac3469bbbe313a894c4d6a2
Certificate Version

Save
 Discard
 Download Public X.509

Updated 2/2/2018 9:55:19 AM

Certificate Identifier

<https://sfdocs.vault.azure.net/certificates/sfdocs-eastus/21ca9a606ac3469bbbe313a894c4d6a2>

Settings

Set activation date?

Activation Date

2018-02-02
9:45:18 AM

(UTC-08:00) --- Current Timezone ---

Set expiration date?

Expiration Date

2019-02-02
9:55:18 AM

(UTC-08:00) --- Current Timezone ---

Enabled? Yes No

Tags

0 tags >

Certificate

Key Identifier

<https://sfdocs.vault.azure.net/keys/sfdocs-eastus/21ca9a606ac3469bbbe313a894c4d6a2>

Secret Identifier

<https://sfdocs.vault.azure.net/secrets/sfdocs-eastus/21ca9a606ac3469bbbe313a894c4d6a2>

X.509 SHA-1 Thumbprint (in hex)

076D2ED43E9032B3AB1B7B7E2AD52BF72EA24161

<< >>

Check the **Configure advanced settings** box to enter client certificates for **admin client** and **read-only client**. In these fields, enter the thumbprint of your admin client certificate and the thumbprint of your read-only user client certificate, if applicable. When administrators attempt to connect to the cluster, they are granted access only if they have a certificate with a thumbprint that matches the thumbprint values entered here.

4. Summary

Now you are ready to deploy the cluster. Before you do that, download the certificate, look inside the large blue informational box for the link. Make sure to keep the cert in a safe place. you need it to connect to your cluster. Since the certificate you downloaded does not have a password, it is advised that you add one.

To complete the cluster creation, click **Create**. You can optionally download the template.

You can see the creation progress in the notifications. (Click the "Bell" icon near the status bar at the upper right of your screen.) If you clicked **Pin to Startboard** while creating the cluster, you see **Deploying Service Fabric Cluster** pinned to the **Start** board. This process will take some time.

In order to perform management operations on your cluster using PowerShell or CLI, you need to connect to your cluster, read more on how to at [connecting to your cluster](#).

View your cluster status

NAME	NODE TYPE	HEALTH STATE	STATUS	UPGRADE DOMAIN	FAULT DOMAIN
_nt1_0	nt1	OK	Up	0	fd:/0
_nt1_1	nt1	OK	Up	1	fd:/1
_nt1_2	nt1	OK	Up	2	fd:/2
_nt1_3	nt1	OK	Up	3	fd:/3
_nt1_4	nt1	OK	Up	4	fd:/4

Once your cluster is created, you can inspect your cluster in the portal:

1. Go to **Browse** and click **Service Fabric Clusters**.
2. Locate your cluster and click it.
3. You can now see the details of your cluster in the dashboard, including the cluster's public endpoint and a link to Service Fabric Explorer.

The **Node Monitor** section on the cluster's dashboard blade indicates the number of VMs that are healthy and not healthy. You can find more details about the cluster's health at [Service Fabric health model introduction](#).

NOTE

Service Fabric clusters require a certain number of nodes to be up always to maintain availability and preserve state - referred to as "maintaining quorum". Therefore, it is typically not safe to shut down all machines in the cluster unless you have first performed a [full backup of your state](#).

Remote connect to a Virtual Machine Scale Set instance or a cluster node

Each of the NodeTypes you specify in your cluster results in a Virtual Machine Scale Set getting set-up.

Next steps

At this point, you have a secure cluster using certificates for management authentication. Next, [connect to your cluster](#) and learn how to [manage application secrets](#). Also, learn about [Service Fabric support options](#).

Continuous deployment to Azure App Service

5/26/2021 • 6 minutes to read • [Edit Online](#)

Azure App Service enables continuous deployment from [GitHub](#), [BitBucket](#), and [Azure Repos](#) repositories by pulling in the latest updates.

NOTE

The **Development Center (Classic)** page in the Azure portal, an earlier version of the deployment experience, was deprecated in March 2021. This change doesn't affect existing deployment settings in your app, and you can continue to manage app deployment in the **Deployment Center** page in the portal.

Prepare your repository

To get automated builds from Azure App Service build server, make sure that your repository root has the correct files in your project.

RUNTIME	ROOT DIRECTORY FILES
ASP.NET (Windows only)	<i>*.sln, *.csproj, or default.aspx</i>
ASP.NET Core	<i>*.sln or *.csproj</i>
PHP	<i>index.php</i>
Ruby (Linux only)	<i>Gemfile</i>
Node.js	<i>server.js, app.js, or package.json</i> with a start script
Python	<i>*.py, requirements.txt, or runtime.txt</i>
HTML	<i>default.htm, default.html, default.asp, index.htm, index.html, or iisstart.htm</i>
WebJobs	<i><job_name>/run.<extension></i> under <i>App_Data/jobs/continuous</i> for continuous WebJobs, or <i>App_Data/jobs/triggered</i> for triggered WebJobs. For more information, see Kudu WebJobs documentation .
Functions	See Continuous deployment for Azure Functions .

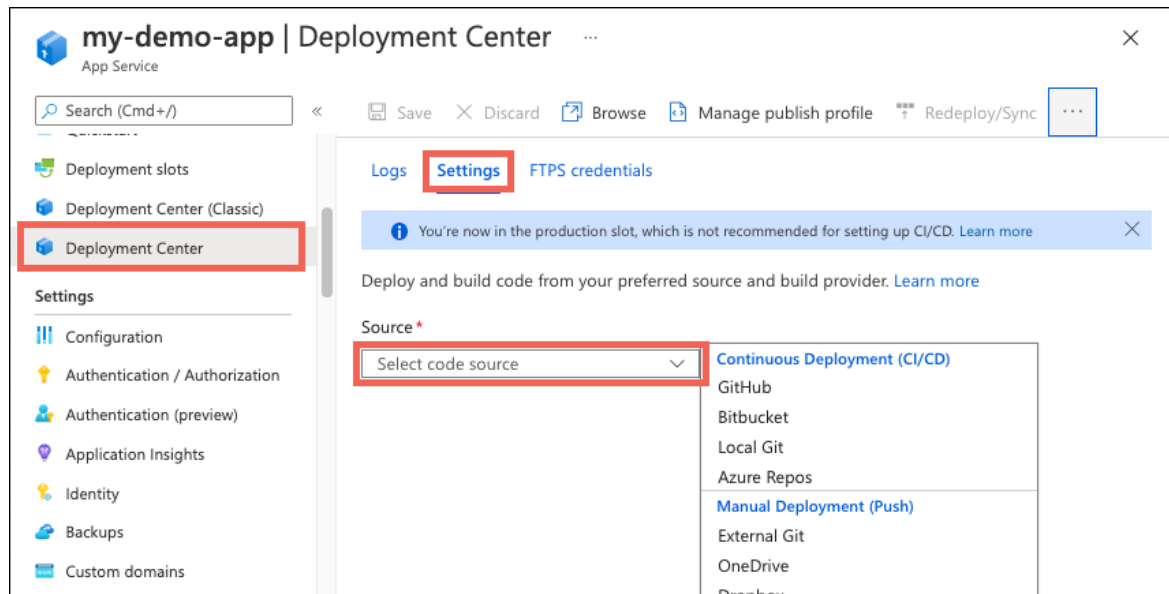
To customize your deployment, include a *.deployment* file in the repository root. For more information, see [Customize deployments](#) and [Custom deployment script](#).

NOTE

If you develop in Visual Studio, let [Visual Studio create a repository for you](#). The project is immediately ready to be deployed by using Git.

Configure deployment source

1. In the [Azure portal](#), navigate to the management page for your App Service app.
2. From the left menu, click **Deployment Center** > **Settings**.
3. In **Source**, select one of the CI/CD options.



Choose the tab that corresponds to your selection for the steps.

- [GitHub](#)
- [BitBucket](#)
- [Local Git](#)
- [Azure Repos](#)

4. [GitHub Actions](#) is the default build provider. To change it, click **Change provider** > **App Service Build Service (Kudu)** > **OK**.

NOTE

To use Azure Pipelines as the build provider for your App Service app, don't configure it in App Service. Instead, configure CI/CD directly from Azure Pipelines. The **Azure Pipelines** option just points you in the right direction.

5. If you're deploying from GitHub for the first time, click **Authorize** and follow the authorization prompts. If you want to deploy from a different user's repository, click **Change Account**.
6. Once you authorize your Azure account with GitHub, select the **Organization**, **Repository**, and **Branch** to configure CI/CD for. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. For more information, see [Managing access to your organization's repositories](#)
7. When GitHub Actions is the chosen build provider, you can select the workflow file you want with the **Runtime stack** and **Version** dropdowns. Azure commits this workflow file into your selected GitHub repository to handle build and deploy tasks. To see the file before saving your changes, click **Preview file**.

NOTE

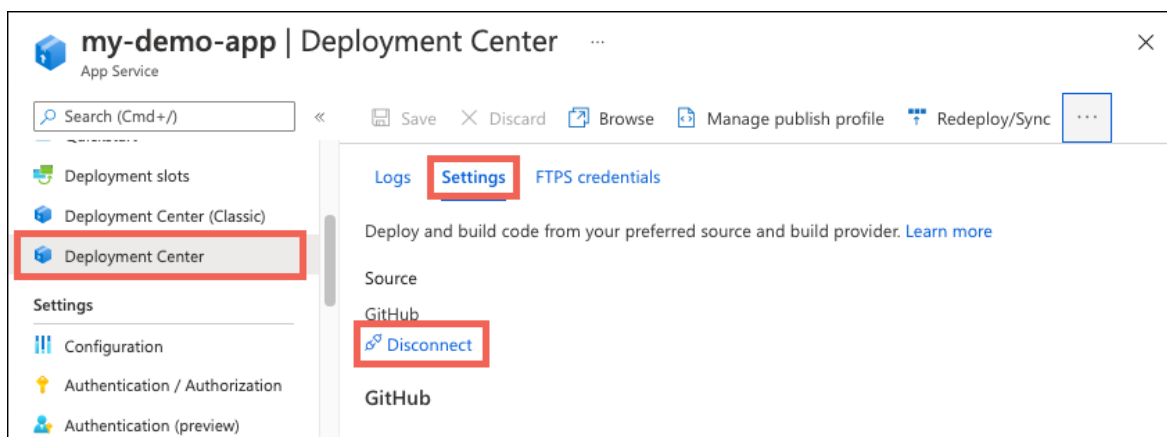
App Service detects the [language stack setting](#) of your app and selects the most appropriate workflow template. If you choose a different template, it may deploy an app that doesn't run properly. For more information, see [How the GitHub Actions build provider works](#).

8. Click **Save**.

New commits in the selected repository and branch now deploy continuously into your App Service app. You can track the commits and deployments in the **Logs** tab.

Disable continuous deployment

1. In the [Azure portal](#), navigate to the management page for your App Service app.
2. From the left menu, click **Deployment Center > Settings > Disconnect**.



3. By default, the GitHub Actions workflow file is preserved in your repository, but it will continue to trigger deployment to your app. To delete it from your repository, select **Delete workflow file**.
4. Click **OK**.

What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. Therefore, the deployment can fail because of locked files. The app may also behave unpredictably during deployment, because not all the files updated at the same time. This is undesirable for a customer-facing app. There are a few different ways to avoid these issues:

- [Run your app from the ZIP package directly](#) without unpacking it.
- Stop your app or enable offline mode for your app during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) enabled.

How the GitHub Actions build provider works

The GitHub Actions build provider is an option for [CI/CD from GitHub](#), and does the following to set up CI/CD:

- Deposits a GitHub Actions workflow file into your GitHub repository to handle build and deploy tasks to App Service.
- Adds the publishing profile for your app as a GitHub secret. The workflow file uses this secret to authenticate with App Service.
- Captures information from the [workflow run logs](#) and displays it in the **Logs** tab in your app's **Deployment**

Center.

You can customize the GitHub Actions build provider in the following ways:

- Customize the workflow file after it's generated in your GitHub repository. For more information, see [Workflow syntax for GitHub Actions](#). Just make sure that the workflow deploys to App Service with the [azure/webapps-deploy](#) action.
- If the selected branch is protected, you can still preview the workflow file without saving the configuration, then manually add it into your repository. This method doesn't give you the log integration with the Azure portal.
- Instead of a publishing profile, deploy using a [service principal](#) in Azure Active Directory.

Authenticate with a service principal

This optional configuration replaces the default authentication with publishing profiles in the generated workflow file.

1. Generate a service principal with the `az ad sp create-for-rbac` command in the [Azure CLI](#). In the following example, replace `<subscription-id>`, `<group-name>`, and `<app-name>` with your own values:

```
az ad sp create-for-rbac --name "myAppDeployAuth" --role contributor \  
    --scopes /subscriptions/<subscription-id>/resourceGroups/<group-  
name>/providers/Microsoft.Web/sites/<app-name> \  
    --sdk-auth
```

IMPORTANT

For security, grant the minimum required access to the service principal. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

2. Save the entire JSON output for the next step, including the top-level `{}`.
3. In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**.
4. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret a name like `AZURE_CREDENTIALS`.
5. In the workflow file generated by the **Deployment Center**, revise the `azure/webapps-deploy` step with code like the following example (which is modified from a Node.js workflow file):

```
- name: Sign in to Azure  
# Use the GitHub secret you added  
- uses: azure/login@v1  
  with:  
    creds: ${{ secrets.AZURE_CREDENTIALS }}  
- name: Deploy to Azure Web App  
# Remove publish-profile  
- uses: azure/webapps-deploy@v2  
  with:  
    app-name: '<app-name>'  
    slot-name: 'production'  
    package: .  
- name: Sign out of Azure  
  run: |  
    az logout
```

Deploy from other repositories

For Windows apps, you can manually configure continuous deployment from a cloud Git or Mercurial repository that the portal doesn't directly support, such as [GitLab](#). You do it by choosing External Git in the Source dropdown. For more information, see [Set up continuous deployment using manual steps](#).

More resources

- [Deploy from Azure Pipelines to Azure App Services](#)
- [Investigate common issues with continuous deployment](#)
- [Use Azure PowerShell](#)
- [Project Kudu](#)

Deploy and remove applications using PowerShell

3/5/2021 • 11 minutes to read • [Edit Online](#)

Once an [application type has been packaged](#), it's ready for deployment into an Azure Service Fabric cluster. Deployment involves the following three steps:

1. Upload the application package to the image store.
2. Register the application type with image store relative path.
3. Create the application instance.

Once the deployed application is no longer required, you can delete the application instance and its application type. To completely remove an application from the cluster involves the following steps:

1. Remove (or delete) the running application instance.
2. Unregister the application type if you no longer need it.
3. Remove the application package from the image store.

If you use Visual Studio for deploying and debugging applications on your local development cluster, all the preceding steps are handled automatically through a PowerShell script. This script is found in the *Scripts* folder of the application project. This article provides background on what that script is doing so that you can perform the same operations outside of Visual Studio.

Another way to deploy an application is by using external provision. The application package can be [packaged as sfpkg](#) and uploaded to an external store. In this case, upload to the image store is not needed. Deployment needs the following steps:

1. Upload the `sfpkg` to an external store. The external store can be any store that exposes a REST http or https endpoint.
2. Register the application type using the external download URI and the application type information.
3. Create the application instance.

For cleanup, remove the application instances and unregister the application type. Because the package was not copied to the image store, there is no temporary location to cleanup. Provisioning from external store is available starting with Service Fabric version 6.1.

NOTE

Visual Studio does not currently support external provision.

Connect to the cluster

Before you run any PowerShell commands in this article, always start by using [Connect-ServiceFabricCluster](#) to connect to the Service Fabric cluster. To connect to the local development cluster, run the following:

```
Connect-ServiceFabricCluster
```

For examples of connecting to a remote cluster or cluster secured using Azure Active Directory, X509 certificates, or Windows Active Directory see [Connect to a secure cluster](#).

Upload the application package

Uploading the application package puts it in a location that's accessible by internal Service Fabric components. If you want to verify the application package locally, use the [Test-ServiceFabricApplicationPackage](#) cmdlet.

The [Copy-ServiceFabricApplicationPackage](#) command uploads the application package to the cluster image store.

Suppose you build and package an application named *MyApplication* in Visual Studio 2015. By default, the application type name listed in the ApplicationManifest.xml is "MyApplicationType". The application package, which contains the necessary application manifest, service manifests, and code/config/data packages, is located in *C:\Users<username>\Documents\Visual Studio 2015\Projects\MyApplication\MyApplication\pkg\Debug*.

The following command lists the contents of the application package:

```
$path = 'C:\Users\<user>\Documents\Visual Studio 2015\Projects\MyApplication\MyApplication\pkg\Debug'  
tree /f $path
```

```
Folder PATH listing for volume OSDisk  
Volume serial number is 0459-2393  
C:\USERS\USER\DOCUMENTS\VISUAL STUDIO 2015\PROJECTS\MYAPPLICATION\MYAPPLICATION\PKG\DEBUG  
| ApplicationManifest.xml  
|  
├── Stateless1Pkg  
|   | ServiceManifest.xml  
|   |  
|   ├── Code  
|   |   | Microsoft.ServiceFabric.Data.dll  
|   |   | Microsoft.ServiceFabric.Data.Interfaces.dll  
|   |   | Microsoft.ServiceFabric.Internal.dll  
|   |   | Microsoft.ServiceFabric.Internal.Strings.dll  
|   |   | Microsoft.ServiceFabric.Services.dll  
|   |   | ServiceFabricServiceModel.dll  
|   |   | Stateless1.exe  
|   |   | Stateless1.exe.config  
|   |   | Stateless1.pdb  
|   |   | System.Fabric.dll  
|   |   | System.Fabric.Strings.dll  
|   |  
|   └── Config  
|       | Settings.xml
```

If the application package is large and/or has many files, you can [compress it](#). The compression reduces the size and the number of files. This results in faster registering and unregistering of the application type. Upload time may be slower currently, especially if you include the time to compress the package.

To compress a package, use the same [Copy-ServiceFabricApplicationPackage](#) command. Compression can be done separate from upload, by using the `SkipCopy` flag, or together with the upload operation. Applying compression on a compressed package is no-op. To uncompress a compressed package, use the same [Copy-ServiceFabricApplicationPackage](#) command with the `UncompressPackage` switch.

The following cmdlet compresses the package without copying it to the image store. The package now includes zipped files for the `Code` and `Config` packages. The application and the service manifests are not zipped, because they are needed for many internal operations (like package sharing, application type name and version extraction for certain validations). Zipping the manifests would make these operations inefficient.

```
Copy-ServiceFabricApplicationPackage -ApplicationPackagePath $path -CompressPackage -SkipCopy  
tree /f $path
```

```

Folder PATH listing for volume OSDisk
Volume serial number is 0459-2393
C:\USERS\USER\DOCUMENTS\VISUAL STUDIO 2015\PROJECTS\MYAPPLICATION\MYAPPLICATION\PKG\DEBUG
|   ApplicationManifest.xml
|
└── Stateless1Pkg
    Code.zip
    Config.zip
    ServiceManifest.xml

```

For large application packages, the compression takes time. For best results, use a fast SSD drive. The compression times and the size of the compressed package also differ based on the package content. For example, here is compression statistics for some packages, which show the initial and the compressed package size, with the compression time.

INITIAL SIZE (MB)	FILE COUNT	COMPRESSION TIME	COMPRESSED PACKAGE SIZE (MB)
100	100	00:00:03.3547592	60
512	100	00:00:16.3850303	307
1024	500	00:00:32.5907950	615
2048	1000	00:01:04.3775554	1231
5012	100	00:02:45.2951288	3074

Once a package is compressed, it can be uploaded to one or multiple Service Fabric clusters as needed. The deployment mechanism is the same for compressed and uncompressed packages. Compressed packages are stored as such in the cluster image store. The packages are uncompressed on the node, before the application is run.

The following example uploads the package to the image store, into a folder named "MyApplicationV1":

```

Copy-ServiceFabricApplicationPackage -ApplicationPackagePath $path -ApplicationPackagePathInImageStore
MyApplicationV1 -TimeoutSec 1800

```

If you do not specify the *-ApplicationPackagePathInImageStore* parameter, the application package is copied into the "Debug" folder in the image store.

NOTE

Copy-ServiceFabricApplicationPackage will automatically detect the appropriate image store connection string if the PowerShell session is connected to a Service Fabric cluster. For Service Fabric versions older than 5.6, the **-ImageStoreConnectionString** argument must be explicitly provided.

```
PS C:\> Copy-ServiceFabricApplicationPackage -ApplicationPackagePath $path -
ApplicationPackagePathInImageStore MyApplicationV1 -ImageStoreConnectionString (Get-
ImageStoreConnectionStringFromClusterManifest(Get-ServiceFabricClusterManifest)) -TimeoutSec 1800
```

The **Get-ImageStoreConnectionStringFromClusterManifest** cmdlet, which is part of the Service Fabric SDK PowerShell module, is used to get the image store connection string. To import the SDK module, run:

```
Import-Module "$ENV:ProgramFiles\Microsoft SDKs\Service
Fabric\Tools\PSModule\ServiceFabricSDK\ServiceFabricSDK.psm1"
```

See [Understand the image store connection string](#) for supplementary information about the image store and image store connection string.

The time it takes to upload a package differs depending on multiple factors. Some of these factors are the number of files in the package, the package size, and the file sizes. The network speed between the source machine and the Service Fabric cluster also impacts the upload time. The default timeout for [Copy-ServiceFabricApplicationPackage](#) is 30 minutes. Depending on the described factors, you may have to increase the timeout. If you are compressing the package in the copy call, you need to also consider the compression time.

Register the application package

The application type and version declared in the application manifest become available for use when the application package is registered. The system reads the package uploaded in the previous step, verifies the package, processes the package contents, and copies the processed package to an internal system location.

Run the [Register-ServiceFabricApplicationType](#) cmdlet to register the application type in the cluster and make it available for deployment:

Register the application package copied to image store

When a package was previously copied to the image store, the register operation specifies the relative path in the image store.

```
Register-ServiceFabricApplicationType -ApplicationPathInImageStore MyApplicationV1
```

```
Register application type succeeded
```

"MyApplicationV1" is the folder in the image store where the application package is located. The application type with name "MyApplicationType" and version "1.0.0" (both are found in the application manifest) is now registered in the cluster.

Register the application package copied to an external store

Starting with Service Fabric version 6.1, provision supports downloading the package from an external store. The download URI represents the path to the `sfpkg` application package from where the application package can be downloaded using HTTP or HTTPS protocols. The package must have been previously uploaded to this external location. The URI must allow READ access so Service Fabric can download the file. The `sfpkg` file must have the extension ".sfpkg". The provision operation should include the application type information, as found in

the application manifest.

```
Register-ServiceFabricApplicationType -ApplicationPackageDownloadUri
"https://sftestresources.blob.core.windows.net:443/sfpkgholder/MyAppPackage.sfpkg" -ApplicationTypeName
MyApp -ApplicationTypeVersion V1 -Async
```

The [Register-ServiceFabricApplicationType](#) command returns only after the system has successfully registered the application package. How long registration takes depends on the size and contents of the application package. If needed, the `-TimeoutSec` parameter can be used to supply a longer timeout (the default timeout is 60 seconds).

If you have a large application package or if you are experiencing timeouts, use the `-Async` parameter. The command returns when the cluster accepts the register command. The register operation continues as needed. The [Get-ServiceFabricApplicationType](#) command lists the application type versions and their registration status. You can use this command to determine when the registration is done.

```
Get-ServiceFabricApplicationType
```

```
ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion    : 1.0.0
Status                   : Available
DefaultParameters       : { "Stateless1_InstanceCount" = "-1" }
```

Remove an application package from the image store

If a package was copied to the image store, you should remove it from the temporary location after the application is successfully registered. Deleting application packages from the image store frees up system resources. Keeping unused application packages consumes disk storage and leads to application performance issues.

```
Remove-ServiceFabricApplicationPackage -ApplicationPackagePathInImageStore MyApplicationV1
```

Create the application

You can instantiate an application from any application type version that has been registered successfully by using the [New-ServiceFabricApplication](#) cmdlet. The name of each application must start with the `"fabric:"` scheme and must be unique for each application instance. Any default services defined in the application manifest of the target application type are also created.

```
New-ServiceFabricApplication fabric:/MyApp MyApplicationType 1.0.0
```

```
ApplicationName          : fabric:/MyApp
ApplicationTypeName       : MyApplicationType
ApplicationTypeVersion    : 1.0.0
ApplicationParameters     : {}
```

Multiple application instances can be created for any given version of a registered application type. Each application instance runs in isolation, with its own work directory and process.

To see which named apps and services are running in the cluster, run the [Get-ServiceFabricApplication](#) and [Get-ServiceFabricService](#) cmdlets:

```
Get-ServiceFabricApplication
```

```
ApplicationName      : fabric:/MyApp
ApplicationTypeName  : MyApplicationType
ApplicationTypeVersion : 1.0.0
ApplicationStatus    : Ready
HealthState         : Ok
ApplicationParameters : {}
```

```
Get-ServiceFabricApplication | Get-ServiceFabricService
```

```
ServiceName          : fabric:/MyApp/Stateless1
ServiceKind          : Stateless
ServiceTypeName      : Stateless1Type
IsServiceGroup       : False
ServiceManifestVersion : 1.0.0
ServiceStatus        : Active
HealthState          : Ok
```

Remove an application

When an application instance is no longer needed, you can permanently remove it by name using the [Remove-ServiceFabricApplication](#) cmdlet. [Remove-ServiceFabricApplication](#) automatically removes all services that belong to the application as well, permanently removing all service state.

WARNING

This operation cannot be reversed, and application state cannot be recovered.

```
Remove-ServiceFabricApplication fabric:/MyApp
```

```
Confirm
Continue with this operation?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"):
Remove application instance succeeded
```

```
Get-ServiceFabricApplication
```

Unregister an application type

When a particular version of an application type is no longer needed, you should unregister the application type using the [Unregister-ServiceFabricApplicationType](#) cmdlet. Unregistering unused application types releases storage space used by the image store by removing the application type files. Unregistering an application type does not remove the application package copied to the image store temporary location, if copy to the image store was used. An application type can be unregistered as long as no applications are instantiated against it and no pending application upgrades are referencing it.

Run [Get-ServiceFabricApplicationType](#) to see the application types currently registered in the cluster:

```
Get-ServiceFabricApplicationType
```

```
ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion   : 1.0.0
Status                   : Available
DefaultParameters       : { "Stateless1_InstanceCount" = "-1" }
```

Run [Unregister-ServiceFabricApplicationType](#) to unregister a specific application type:

```
Unregister-ServiceFabricApplicationType MyApplicationType 1.0.0
```

Troubleshooting

Copy-ServiceFabricApplicationPackage asks for an ImageStoreConnectionString

The Service Fabric SDK environment should already have the correct defaults set up. But if needed, the `ImageStoreConnectionString` for all commands should match the value that the Service Fabric cluster is using. You can find the `ImageStoreConnectionString` in the cluster manifest, retrieved using the [Get-ServiceFabricClusterManifest](#) and `Get-ImageStoreConnectionStringFromClusterManifest` commands:

```
Get-ImageStoreConnectionStringFromClusterManifest(Get-ServiceFabricClusterManifest)
```

The `Get-ImageStoreConnectionStringFromClusterManifest` cmdlet, which is part of the Service Fabric SDK PowerShell module, is used to get the image store connection string. To import the SDK module, run:

```
Import-Module "$ENV:ProgramFiles\Microsoft SDKs\Service
Fabric\Tools\PSModule\ServiceFabricSDK\ServiceFabricSDK.psm1"
```

The `ImageStoreConnectionString` is found in the cluster manifest:

```
<ClusterManifest xmlns:xsd="https://www.w3.org/2001/XMLSchema" xmlns:xsi="https://www.w3.org/2001/XMLSchema-
instance" Name="Server-Default-SingleNode" Version="1.0"
xmlns="http://schemas.microsoft.com/2011/01/fabric">

  [...]

  <Section Name="Management">
    <Parameter Name="ImageStoreConnectionString" Value="file:D:\ServiceFabric\Data\ImageStore" />
  </Section>

  [...]
```

See [Understand the image store connection string](#) for supplementary information about the image store and image store connection string.

Deploy large application package

Issue: [Copy-ServiceFabricApplicationPackage](#) times out for a large application package (order of GB). Try:

- Specify a larger timeout for [Copy-ServiceFabricApplicationPackage](#) command, with `TimeoutSec` parameter. By default, the timeout is 30 minutes.
- Check the network connection between your source machine and cluster. If the connection is slow, consider using a machine with a better network connection. If the client machine is in another region than the cluster, consider using a client machine in a closer or same region as the cluster.

- Check if you are hitting external throttling. For example, when the image store is configured to use azure storage, upload may be throttled.

Issue: Upload package completed successfully, but [Register-ServiceFabricApplicationType](#) times out. Try:

- [Compress the package](#) before copying to the image store. The compression reduces the size and the number of files, which in turn reduces the amount of traffic and work that Service Fabric must perform. The upload operation may be slower (especially if you include the compression time), but register and un-register the application type are faster.
- Specify a larger timeout for [Register-ServiceFabricApplicationType](#) with `TimeoutSec` parameter.
- Specify `Async` switch for [Register-ServiceFabricApplicationType](#). The command returns when the cluster accepts the command and the registration of the application type continues asynchronously. For this reason, there is no need to specify a higher timeout in this case. The [Get-ServiceFabricApplicationType](#) command lists all successfully registered application type versions and their registration status. You can use this command to determine when the registration is done.

```
Get-ServiceFabricApplicationType
```

```
ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion    : 1.0.0
Status                   : Available
DefaultParameters        : { "Stateless1_InstanceCount" = "-1" }
```

Deploy application package with many files

Issue: [Register-ServiceFabricApplicationType](#) times out for an application package with many files (order of thousands). Try:

- [Compress the package](#) before copying to the image store. The compression reduces the number of files.
- Specify a larger timeout for [Register-ServiceFabricApplicationType](#) with `TimeoutSec` parameter.
- Specify `Async` switch for [Register-ServiceFabricApplicationType](#). The command returns when the cluster accepts the command and the registration of the application type continues asynchronously. For this reason, there is no need to specify a higher timeout in this case. The [Get-ServiceFabricApplicationType](#) command lists all successfully registered application type versions and their registration status. You can use this command to determine when the registration is done.

```
Get-ServiceFabricApplicationType
```

```
ApplicationTypeName      : MyApplicationType
ApplicationTypeVersion    : 1.0.0
Status                   : Available
DefaultParameters        : { "Stateless1_InstanceCount" = "-1" }
```

Next steps

[Package an application](#)

[Service Fabric application upgrade](#)

[Service Fabric health introduction](#)

[Diagnose and troubleshoot a Service Fabric service](#)

Model an application in Service Fabric

Tutorial: Create and Manage Linux VMs with the Azure CLI

4/22/2021 • 8 minutes to read • [Edit Online](#)

Azure virtual machines provide a fully configurable and flexible computing environment. This tutorial covers basic Azure virtual machine deployment items such as selecting a VM size, selecting a VM image, and deploying a VM. You learn how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

This tutorial uses the CLI within the [Azure Cloud Shell](#), which is constantly updated to the latest version. To open the Cloud Shell, select **Try it** from the top of any code block.

If you choose to install and use the CLI locally, this tutorial requires that you are running the Azure CLI version 2.0.30 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Create resource group

Create a resource group with the `az group create` command.

An Azure resource group is a logical container into which Azure resources are deployed and managed. A resource group must be created before a virtual machine. In this example, a resource group named `myResourceGroupVM` is created in the `eastus` region.

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

The resource group is specified when creating or modifying a VM, which can be seen throughout this tutorial.

Create virtual machine

Create a virtual machine with the `az vm create` command.

When you create a virtual machine, several options are available such as operating system image, disk sizing, and administrative credentials. The following example creates a VM named `myVM` that runs Ubuntu Server. A user account named `azureuser` is created on the VM, and SSH keys are generated if they do not exist in the default key location (`~/ssh`):

```
az vm create \  
  --resource-group myResourceGroupVM \  
  --name myVM \  
  --image UbuntuLTS \  
  --admin-username azureuser \  
  --generate-ssh-keys
```

It may take a few minutes to create the VM. Once the VM has been created, the Azure CLI outputs information about the VM. Take note of the `publicIpAddress`, this address can be used to access the virtual machine..

```
{
  "fqdns": "",
  "id": "/subscriptions/d5b9d4b7-6fc1-0000-0000-000000000000/resourceGroups/myResourceGroupVM/providers/Microsoft.Compute/virtualMachines/myVM",
  "location": "eastus",
  "macAddress": "00-0D-3A-23-9A-49",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "52.174.34.95",
  "resourceGroup": "myResourceGroupVM"
}
```

Connect to VM

You can now connect to the VM with SSH in the Azure Cloud Shell or from your local computer. Replace the example IP address with the `publicIpAddress` noted in the previous step.

```
ssh azureuser@52.174.34.95
```

Once logged in to the VM, you can install and configure applications. When you are finished, you close the SSH session as normal:

```
exit
```

Understand VM images

The Azure marketplace includes many images that can be used to create VMs. In the previous steps, a virtual machine was created using an Ubuntu image. In this step, the Azure CLI is used to search the marketplace for a CentOS image, which is then used to deploy a second virtual machine.

To see a list of the most commonly used images, use the [az vm image list](#) command.

```
az vm image list --output table
```

The command output returns the most popular VM images on Azure.

Offer UrnAlias	Publisher Version	Skus	Urn
WindowsServer Datacenter:latest	MicrosoftWindowsServer Win2016Datacenter	2016-Datacenter latest	MicrosoftWindowsServer:WindowsServer:2016-
WindowsServer Datacenter:latest	MicrosoftWindowsServer Win2012R2Datacenter	2012-R2-Datacenter latest	MicrosoftWindowsServer:WindowsServer:2012-R2-
WindowsServer SP1:latest	MicrosoftWindowsServer Win2008R2SP1	2008-R2-SP1 latest	MicrosoftWindowsServer:WindowsServer:2008-R2-
WindowsServer Datacenter:latest	MicrosoftWindowsServer Win2012Datacenter	2012-Datacenter latest	MicrosoftWindowsServer:WindowsServer:2012-
UbuntuServer UbuntuLTS	Canonical latest	16.04-LTS	Canonical:UbuntuServer:16.04-LTS:latest
CentOS	OpenLogic	7.3	OpenLogic:CentOS:7.3:latest
CentOS	latest		
openSUSE-Leap	SUSE	42.2	SUSE:openSUSE-Leap:42.2:latest
openSUSE-Leap	latest		
RHEL	RedHat	7.3	RedHat:RHEL:7.3:latest
RHEL	latest		
SLES	SUSE	12-SP2	SUSE:SLES:12-SP2:latest
SLES	latest		
Debian	credativ	8	credativ:Debian:8:latest
Debian	latest		
CoreOS	CoreOS	Stable	CoreOS:CoreOS:Stable:latest
CoreOS	latest		

A full list can be seen by adding the `--all` argument. The image list can also be filtered by `--publisher` or `--offer`. In this example, the list is filtered for all images with an offer that matches *CentOS*.

```
az vm image list --offer CentOS --all --output table
```

Partial output:

Offer	Publisher	Skus	Urn	Version
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.201501	6.5.201501
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.201503	6.5.201503
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.201506	6.5.201506
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.20150904	6.5.20150904
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.20160309	6.5.20160309
CentOS	OpenLogic	6.5	OpenLogic:CentOS:6.5:6.5.20170207	6.5.20170207

To deploy a VM using a specific image, take note of the value in the *Urn* column, which consists of the publisher, offer, SKU, and optionally a version number to [identify](#) the image. When specifying the image, the image version number can be replaced with "latest", which selects the latest version of the distribution. In this example, the `--image` argument is used to specify the latest version of a CentOS 6.5 image.

```
az vm create --resource-group myResourceGroupVM --name myVM2 --image OpenLogic:CentOS:6.5:latest --generate-ssh-keys
```

Understand VM sizes

A virtual machine size determines the amount of compute resources such as CPU, GPU, and memory that are made available to the virtual machine. Virtual machines need to be sized appropriately for the expected workload. If workload increases, an existing virtual machine can be resized.

VM Sizes

The following table categorizes sizes into use cases.

TYPE	COMMON SIZES	DESCRIPTION
General purpose	B, Dsv3, Dv3, DSv2, Dv2, Av2, DC	Balanced CPU-to-memory. Ideal for dev / test and small to medium applications and data solutions.
Compute optimized	Fsv2	High CPU-to-memory. Good for medium traffic applications, network appliances, and batch processes.
Memory optimized	Esv3, Ev3, M, DSv2, Dv2	High memory-to-core. Great for relational databases, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2, Ls	High disk throughput and IO. Ideal for Big Data, SQL, and NoSQL databases.
GPU	NV, NVv2, NC, NCv2, NCv3, ND	Specialized VMs targeted for heavy graphic rendering and video editing.
High performance	H	Our most powerful CPU VMs with optional high-throughput network interfaces (RDMA).

Find available VM sizes

To see a list of VM sizes available in a particular region, use the [az vm list-sizes](#) command.

```
az vm list-sizes --location eastus --output table
```

Partial output:

MaxDataDiskCount	MemoryInMb	Name	NumberOfCores	OsDiskSizeInMb
ResourceDiskSizeInMb				
-----	-----	-----	-----	-----

7168	2	3584 Standard_DS1	1	1047552
14336	4	7168 Standard_DS2	2	1047552
28672	8	14336 Standard_DS3	4	1047552
57344	16	28672 Standard_DS4	8	1047552
28672	4	14336 Standard_DS11	2	1047552
57344	8	28672 Standard_DS12	4	1047552
114688	16	57344 Standard_DS13	8	1047552
229376	32	114688 Standard_DS14	16	1047552
20480	1	768 Standard_A0	1	1047552
71680	2	1792 Standard_A1	1	1047552
138240	4	3584 Standard_A2	2	1047552
291840	8	7168 Standard_A3	4	1047552
138240	4	14336 Standard_A5	2	1047552
619520	16	14336 Standard_A4	8	1047552
291840	8	28672 Standard_A6	4	1047552
619520	16	57344 Standard_A7	8	1047552

Create VM with specific size

In the previous VM creation example, a size was not provided, which results in a default size. A VM size can be selected at creation time using `az vm create` and the `--size` argument.

```
az vm create \
  --resource-group myResourceGroupVM \
  --name myVM3 \
  --image UbuntuLTS \
  --size Standard_F4s \
  --generate-ssh-keys
```

Resize a VM

After a VM has been deployed, it can be resized to increase or decrease resource allocation. You can view the current of size of a VM with `az vm show`:

```
az vm show --resource-group myResourceGroupVM --name myVM --query hardwareProfile.vmSize
```

Before resizing a VM, check if the desired size is available on the current Azure cluster. The `az vm list-vm-resize-options` command returns the list of sizes.

```
az vm list-vm-resize-options --resource-group myResourceGroupVM --name myVM --query [].name
```

If the desired size is available, the VM can be resized from a powered-on state, however it is rebooted during the operation. Use the [az vm resize](#) command to perform the resize.

```
az vm resize --resource-group myResourceGroupVM --name myVM --size Standard_DS4_v2
```

If the desired size is not on the current cluster, the VM needs to be deallocated before the resize operation can occur. Use the [az vm deallocate](#) command to stop and deallocate the VM. Note, when the VM is powered back on, any data on the temp disk may be removed. The public IP address also changes unless a static IP address is being used.

```
az vm deallocate --resource-group myResourceGroupVM --name myVM
```

Once deallocated, the resize can occur.

```
az vm resize --resource-group myResourceGroupVM --name myVM --size Standard_GS1
```

After the resize, the VM can be started.

```
az vm start --resource-group myResourceGroupVM --name myVM
```

VM power states

An Azure VM can have one of many power states. This state represents the current state of the VM from the standpoint of the hypervisor.

Power states

POWER STATE	DESCRIPTION
Starting	Indicates the virtual machine is being started.
Running	Indicates that the virtual machine is running.
Stopping	Indicates that the virtual machine is being stopped.
Stopped	Indicates that the virtual machine is stopped. Virtual machines in the stopped state still incur compute charges.
Deallocating	Indicates that the virtual machine is being deallocated.
Deallocated	Indicates that the virtual machine is removed from the hypervisor but still available in the control plane. Virtual machines in the Deallocated state do not incur compute charges.
-	Indicates that the power state of the virtual machine is unknown.

Find the power state

To retrieve the state of a particular VM, use the [az vm get-instance-view](#) command. Be sure to specify a valid name for a virtual machine and resource group.

```
az vm get-instance-view \  
  --name myVM \  
  --resource-group myResourceGroupVM \  
  --query instanceView.statuses[1] --output table
```

Output:

```
ode           DisplayStatus   Level  
-----  
PowerState/running VM running   Info
```

To retrieve the power state of all the VMs in your subscription, use the [Virtual Machines - List All API](#) with parameter `statusOnly` set to `true`.

Management tasks

During the life-cycle of a virtual machine, you may want to run management tasks such as starting, stopping, or deleting a virtual machine. Additionally, you may want to create scripts to automate repetitive or complex tasks. Using the Azure CLI, many common management tasks can be run from the command line or in scripts.

Get IP address

This command returns the private and public IP addresses of a virtual machine.

```
az vm list-ip-addresses --resource-group myResourceGroupVM --name myVM --output table
```

Stop virtual machine

```
az vm stop --resource-group myResourceGroupVM --name myVM
```

Start virtual machine

```
az vm start --resource-group myResourceGroupVM --name myVM
```

Delete resource group

Deleting a resource group also deletes all resources contained within, such as the VM, virtual network, and disk.

The `--no-wait` parameter returns control to the prompt without waiting for the operation to complete. The

`--yes` parameter confirms that you wish to delete the resources without an additional prompt to do so.

```
az group delete --name myResourceGroupVM --no-wait --yes
```

Next steps

In this tutorial, you learned about basic VM creation and management such as how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

Advance to the next tutorial to learn about VM disks.

Tutorial: Create and Manage Windows VMs with Azure PowerShell

3/10/2021 • 7 minutes to read • [Edit Online](#)

Azure virtual machines provide a fully configurable and flexible computing environment. This tutorial covers basic Azure virtual machine (VM) deployment tasks like selecting a VM size, selecting a VM image, and deploying a VM. You learn how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

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.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/powershell>. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

Create resource group

Create a resource group with the [New-AzResourceGroup](#) command.

An Azure resource group is a logical container into which Azure resources are deployed and managed. A resource group must be created before a virtual machine. In the following example, a resource group named *myResourceGroupVM* is created in the *EastUS* region:

```
New-AzResourceGroup `
  -ResourceGroupName "myResourceGroupVM" `
  -Location "EastUS"
```

The resource group is specified when creating or modifying a VM, which can be seen throughout this tutorial.

Create a VM

When creating a VM, several options are available like operating system image, network configuration, and administrative credentials. This example creates a VM named *myVM*, running the default version of Windows Server 2016 Datacenter.

Set the username and password needed for the administrator account on the VM with [Get-Credential](#):

```
$cred = Get-Credential
```

Create the VM with [New-AzVM](#).

```
New-AzVm `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" `
  -Location "EastUS" `
  -VirtualNetworkName "myVnet" `
  -SubnetName "mySubnet" `
  -SecurityGroupName "myNetworkSecurityGroup" `
  -PublicIpAddressName "myPublicIpAddress" `
  -Credential $cred
```

Connect to VM

After the deployment has completed, create a remote desktop connection with the VM.

Run the following commands to return the public IP address of the VM. Take note of this IP Address so you can connect to it with your browser to test web connectivity in a future step.

```
Get-AzPublicIpAddress `
  -ResourceGroupName "myResourceGroupVM" | Select IPAddress
```

Use the following command, on your local machine, to create a remote desktop session with the VM. Replace the IP address with the *publicIpAddress* of your VM. When prompted, enter the credentials used when creating the VM.

```
mstsc /v:<publicIpAddress>
```

In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username and password you created for the VM and then click **OK**.

Understand marketplace images

The Azure marketplace includes many images that can be used to create a new VM. In the previous steps, a VM was created using the Windows Server 2016 Datacenter image. In this step, the PowerShell module is used to search the marketplace for other Windows images, which can also be used as a base for new VMs. This process consists of finding the publisher, offer, SKU, and optionally a version number to **identify** the image.

Use the [Get-AzVMImagePublisher](#) command to return a list of image publishers:

```
Get-AzVMImagePublisher -Location "EastUS"
```

Use the [Get-AzVMImageOffer](#) to return a list of image offers. With this command, the returned list is filtered on the specified publisher named `MicrosoftWindowsServer`:

```
Get-AzVMImageOffer `
  -Location "EastUS" `
  -PublisherName "MicrosoftWindowsServer"
```

The results will look something like this example:

Offer	PublisherName	Location
Windows-HUB	MicrosoftWindowsServer	EastUS
WindowsServer	MicrosoftWindowsServer	EastUS
WindowsServer-HUB	MicrosoftWindowsServer	EastUS

The `Get-AzVMImageSku` command will then filter on the publisher and offer name to return a list of image names.

```
Get-AzVMImageSku `
  -Location "EastUS" `
  -PublisherName "MicrosoftWindowsServer" `
  -Offer "WindowsServer"
```

The results will look something like this example:

Skus	Offer	PublisherName	Location
2008-R2-SP1	WindowsServer	MicrosoftWindowsServer	EastUS
2008-R2-SP1-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2012-Datacenter	WindowsServer	MicrosoftWindowsServer	EastUS
2012-Datacenter-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2012-R2-Datacenter	WindowsServer	MicrosoftWindowsServer	EastUS
2012-R2-Datacenter-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-Server-Core	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-Server-Core-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-with-Containers	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-with-Containers-smalldisk	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Datacenter-with-RDSH	WindowsServer	MicrosoftWindowsServer	EastUS
2016-Nano-Server	WindowsServer	MicrosoftWindowsServer	EastUS

This information can be used to deploy a VM with a specific image. This example deploys a VM using the latest version of a Windows Server 2016 with Containers image.

```
New-AzVm `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM2" `
  -Location "EastUS" `
  -VirtualNetworkName "myVnet" `
  -SubnetName "mySubnet" `
  -SecurityGroupName "myNetworkSecurityGroup" `
  -PublicIpAddressName "myPublicIpAddress2" `
  -ImageName "MicrosoftWindowsServer:WindowsServer:2016-Datacenter-with-Containers:latest" `
  -Credential $cred `
  -AsJob
```

The `-AsJob` parameter creates the VM as a background task, so the PowerShell prompts return to you. You can view details of background jobs with the `Get-Job` cmdlet.

Understand VM sizes

The VM size determines the amount of compute resources like CPU, GPU, and memory that are made available to the VM. Virtual machines should be created using a VM size appropriate for the workload. If a workload increases, an existing virtual machine can also be resized.

VM Sizes

The following table categorizes sizes into use cases.

TYPE	COMMON SIZES	DESCRIPTION
General purpose	B, Dsv3, Dv3, DSv2, Dv2, Av2, DC	Balanced CPU-to-memory. Ideal for dev / test and small to medium applications and data solutions.
Compute optimized	Fsv2	High CPU-to-memory. Good for medium traffic applications, network appliances, and batch processes.
Memory optimized	Esv3, Ev3, M, DSv2, Dv2	High memory-to-core. Great for relational databases, medium to large caches, and in-memory analytics.
Storage optimized	Lsv2, Ls	High disk throughput and IO. Ideal for Big Data, SQL, and NoSQL databases.
GPU	NV, NVv2, NC, NCv2, NCv3, ND	Specialized VMs targeted for heavy graphic rendering and video editing.
High performance	H	Our most powerful CPU VMs with optional high-throughput network interfaces (RDMA).

Find available VM sizes

To see a list of VM sizes available in a particular region, use the [Get-AzVMSize](#) command.

```
Get-AzVMSize -Location "EastUS"
```

Resize a VM

After a VM has been deployed, it can be resized to increase or decrease resource allocation.

Before resizing a VM, check if the size you want is available on the current VM cluster. The [Get-AzVMSize](#) command returns a list of sizes.

```
Get-AzVMSize -ResourceGroupName "myResourceGroupVM" -VMName "myVM"
```

If the size is available, the VM can be resized from a powered-on state, however it is rebooted during the operation.

```
$vm = Get-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -VMName "myVM"
$vm.HardwareProfile.VmSize = "Standard_DS3_v2"
Update-AzVM `
  -VM $vm `
  -ResourceGroupName "myResourceGroupVM"
```

If the size you want isn't available on the current cluster, the VM needs to be deallocated before the resize operation can occur. Deallocating a VM will remove any data on the temp disk, and the public IP address will change unless a static IP address is being used.

```

Stop-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" -Force
$vm = Get-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -VMName "myVM"
$vm.HardwareProfile.VmSize = "Standard_E2s_v3"
Update-AzVM -VM $vm `
  -ResourceGroupName "myResourceGroupVM"
Start-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name $vm.name

```

VM power states

An Azure VM can have one of many power states.

POWER STATE	DESCRIPTION
Starting	The virtual machine is being started.
Running	The virtual machine is running.
Stopping	The virtual machine is being stopped.
Stopped	The VM is stopped. Virtual machines in the stopped state still incur compute charges.
Deallocating	The VM is being deallocated.
Deallocated	Indicates that the VM is removed from the hypervisor but is still available in the control plane. Virtual machines in the <code>Deallocated</code> state do not incur compute charges.
-	The power state of the VM is unknown.

To get the state of a particular VM, use the [Get-AzVM](#) command. Be sure to specify a valid name for a VM and resource group.

```

Get-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" `
  -Status | Select @{n="Status"; e={$_.Statuses[1].Code}}

```

The output will look something like this example:

```

Status
-----
PowerState/running

```

To retrieve the power state of all the VMs in your subscription, use the [Virtual Machines - List All API](#) with parameter `statusOnly` set to `true`.

Management tasks

During the lifecycle of a VM, you may want to run management tasks like starting, stopping, or deleting a VM. Additionally, you may want to create scripts to automate repetitive or complex tasks. Using Azure PowerShell, many common management tasks can be run from the command line or in scripts.

Stop a VM

Stop and deallocate a VM with [Stop-AzVM](#):

```
Stop-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM" -Force
```

If you want to keep the VM in a provisioned state, use the `-StayProvisioned` parameter.

Start a VM

```
Start-AzVM `
  -ResourceGroupName "myResourceGroupVM" `
  -Name "myVM"
```

Delete resource group

Everything inside of a resource group is deleted when you delete the resource group.

```
Remove-AzResourceGroup `
  -Name "myResourceGroupVM" `
  -Force
```

Next steps

In this tutorial, you learned about basic VM creation and management such as how to:

- Create and connect to a VM
- Select and use VM images
- View and use specific VM sizes
- Resize a VM
- View and understand VM state

Advance to the next tutorial to learn about VM disks.

[Create and Manage VM disks](#)

Quickstart: Azure Blob Storage client library v12 for .NET

4/22/2021 • 8 minutes to read • [Edit Online](#)

Get started with the Azure Blob Storage client library v12 for .NET. Azure Blob Storage is Microsoft's object storage solution for the cloud. Follow steps to install the package and try out example code for basic tasks. Blob storage is optimized for storing massive amounts of unstructured data.

Use the Azure Blob Storage client library v12 for .NET to:

- Create a container
- Upload a blob to Azure Storage
- List all of the blobs in a container
- Download the blob to your local computer
- Delete a container

Additional resources:

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\)](#)
- [Samples](#)

NOTE

The features described in this article are also available to accounts that have a hierarchical namespace. To review limitations, see the [Blob storage features available in Azure Data Lake Storage Gen2](#) article.

Prerequisites

- Azure subscription - [create one for free](#)
- Azure storage account - [create a storage account](#)
- Current [.NET Core SDK](#) for your operating system. Be sure to get the SDK and not the runtime.

Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library v12 for .NET.

Create the project

Create a .NET Core application named *BlobQuickstartV12*.

1. In a console window (such as cmd, PowerShell, or Bash), use the `dotnet new` command to create a new console app with the name *BlobQuickstartV12*. This command creates a simple "Hello World" C# project with a single source file: *Program.cs*.

```
dotnet new console -n BlobQuickstartV12
```


2. Switch to the newly created *BlobQuickstartV12* directory.

```
cd BlobQuickstartV12
```

3. In side the *BlobQuickstartV12* directory, create another directory called *data*. This is where the blob data files will be created and stored.

```
mkdir data
```

Install the package

While still in the application directory, install the Azure Blob Storage client library for .NET package by using the `dotnet add package` command.

```
dotnet add package Azure.Storage.Blobs
```

Set up the app framework

From the project directory:

1. Open the *Program.cs* file in your editor.
2. Remove the `Console.WriteLine("Hello World!");` statement.
3. Add `using` directives.
4. Update the `Main` method declaration to support async.

Here's the code:

```
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using System;
using System.IO;
using System.Threading.Tasks;

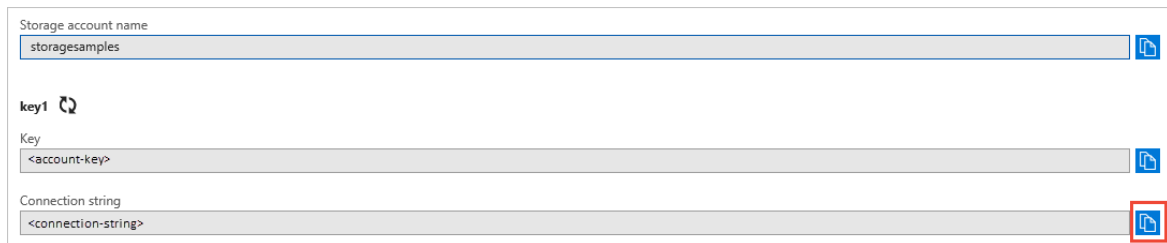
namespace BlobQuickstartV12
{
    class Program
    {
        static async Task Main()
        {
        }
    }
}
```

Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. View your storage account credentials by following these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the **Security + networking** section of the storage account overview, select **Access keys**. Here, you can view your account access keys and the complete connection string for each key.
4. Find the **Connection string** value under **key1**, and select the **Copy** button to copy the connection

string. You will add the connection string value to an environment variable in the next step.



Configure your storage connection string

After you have copied your connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace `<yourconnectionstring>` with your actual connection string.

Windows

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

Linux

```
export AZURE_STORAGE_CONNECTION_STRING="<yourconnectionstring>"
```

macOS

```
export AZURE_STORAGE_CONNECTION_STRING="<yourconnectionstring>"
```

Restart programs

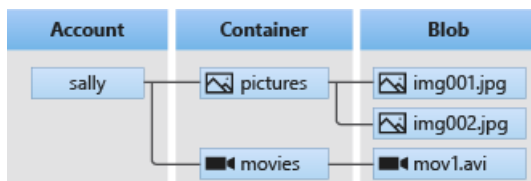
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

Object model

Azure Blob Storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that does not adhere to a particular data model or definition, such as text or binary data. Blob storage offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use the following .NET classes to interact with these resources:

- **BlobServiceClient**: The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.
- **BlobContainerClient**: The `BlobContainerClient` class allows you to manipulate Azure Storage containers and their blobs.

- **BlobClient**: The `BlobClient` class allows you to manipulate Azure Storage blobs.
- **BlobDownloadInfo**: The `BlobDownloadInfo` class represents the properties and content returned from downloading a blob.

Code examples

These example code snippets show you how to perform the following with the Azure Blob Storage client library for .NET:

- [Get the connection string](#)
- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

Get the connection string

The code below retrieves the connection string for the storage account from the environment variable created in the [Configure your storage connection string](#) section.

Add this code inside the `Main` method:

```
Console.WriteLine("Azure Blob Storage v12 - .NET quickstart sample\n");

// Retrieve the connection string for use with the application. The storage
// connection string is stored in an environment variable on the machine
// running the application called AZURE_STORAGE_CONNECTION_STRING. If the
// environment variable is created after the application is launched in a
// console or with Visual Studio, the shell or application needs to be closed
// and reloaded to take the environment variable into account.
string connectionString = Environment.GetEnvironmentVariable("AZURE_STORAGE_CONNECTION_STRING");
```

Create a container

Decide on a name for the new container. The code below appends a GUID value to the container name to ensure that it is unique.

IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Create an instance of the `BlobServiceClient` class. Then, call the `CreateBlobContainerAsync` method to create the container in your storage account.

Add this code to the end of the `Main` method:

```
// Create a BlobServiceClient object which will be used to create a container client
BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

//Create a unique name for the container
string containerName = "quickstartblobs" + Guid.NewGuid().ToString();

// Create the container and return a container client object
BlobContainerClient containerClient = await blobServiceClient.CreateBlobContainerAsync(containerName);
```

Upload blobs to a container

The following code snippet:

1. Creates a text file in the local *data* directory.
2. Gets a reference to a [BlobClient](#) object by calling the [GetBlobClient](#) method on the container from the [Create a container](#) section.
3. Uploads the local text file to the blob by calling the [UploadAsync](#) method. This method creates the blob if it doesn't already exist, and overwrites it if it does.

Add this code to the end of the `Main` method:

```
// Create a local file in the ./data/ directory for uploading and downloading
string localPath = "./data/";
string fileName = "quickstart" + Guid.NewGuid().ToString() + ".txt";
string localFilePath = Path.Combine(localPath, fileName);

// Write text to the file
await File.WriteAllTextAsync(localFilePath, "Hello, World!");

// Get a reference to a blob
BlobClient blobClient = containerClient.GetBlobClient(fileName);

Console.WriteLine("Uploading to Blob storage as blob:\n\t {0}\n", blobClient.Uri);

// Open the file and upload its data
using FileStream uploadFileStream = File.OpenRead(localFilePath);
await blobClient.UploadAsync(uploadFileStream, true);
uploadFileStream.Close();
```

List the blobs in a container

List the blobs in the container by calling the [GetBlobsAsync](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

Add this code to the end of the `Main` method:

```
Console.WriteLine("Listing blobs...");

// List all blobs in the container
await foreach (BlobItem blobItem in containerClient.GetBlobsAsync())
{
    Console.WriteLine("\t" + blobItem.Name);
}
```

Download blobs

Download the previously created blob by calling the [DownloadAsync](#) method. The example code adds a suffix of "DOWNLOADED" to the file name so that you can see both files in local file system.

Add this code to the end of the `Main` method:

```

// Download the blob to a local file
// Append the string "DOWNLOADED" before the .txt extension
// so you can compare the files in the data directory
string downloadFilePath = localFilePath.Replace(".txt", "DOWNLOADED.txt");

Console.WriteLine("\nDownloading blob to\n\t{0}\n", downloadFilePath);

// Download the blob's contents and save it to a file
BlobDownloadInfo download = await blobClient.DownloadAsync();

using (FileStream downloadFileStream = File.OpenWrite(downloadFilePath))
{
    await download.Content.CopyToAsync(downloadFileStream);
    downloadFileStream.Close();
}

```

Delete a container

The following code cleans up the resources the app created by deleting the entire container by using [DeleteAsync](#). It also deletes the local files created by the app.

The app pauses for user input by calling `Console.ReadLine` before it deletes the blob, container, and local files. This is a good chance to verify that the resources were actually created correctly, before they are deleted.

Add this code to the end of the `Main` method:

```

// Clean up
Console.Write("Press any key to begin clean up");
Console.ReadLine();

Console.WriteLine("Deleting blob container...");
await containerClient.DeleteAsync();

Console.WriteLine("Deleting the local source and downloaded files...");
File.Delete(localFilePath);
File.Delete(downloadFilePath);

Console.WriteLine("Done");

```

Run the code

This app creates a test file in your local *data* folder and uploads it to Blob storage. The example then lists the blobs in the container and downloads the file with a new name so that you can compare the old and new files.

Navigate to your application directory, then build and run the application.

```
dotnet build
```

```
dotnet run
```

The output of the app is similar to the following example:

```
Azure Blob Storage v12 - .NET quickstart sample

Uploading to Blob storage as blob:
  https://mystorageacct.blob.core.windows.net/quickstartblobs60c70d78-8d93-43ae-954d-
8322058cfd64/quickstart2fe6c5b4-7918-46cb-96f4-8c4c5cb2fd31.txt

Listing blobs...
  quickstart2fe6c5b4-7918-46cb-96f4-8c4c5cb2fd31.txt

Downloading blob to
  ./data/quickstart2fe6c5b4-7918-46cb-96f4-8c4c5cb2fd31DOWNLOADED.txt

Press any key to begin clean up
Deleting blob container...
Deleting the local source and downloaded files...
Done
```

Before you begin the clean up process, check your *data* folder for the two files. You can open them and observe that they are identical.

After you've verified the files, press the **Enter** key to delete the test files and finish the demo.

Next steps

In this quickstart, you learned how to upload, download, and list blobs using .NET.

To see Blob storage sample apps, continue to:

[Azure Blob Storage SDK v12 .NET samples](#)

- For tutorials, samples, quick starts and other documentation, visit [Azure for .NET and .NET Core developers](#).
- To learn more about .NET Core, see [Get started with .NET in 10 minutes](#).

Develop for Azure Files with .NET

6/15/2021 • 22 minutes to read • [Edit Online](#)

Learn the basics of developing .NET applications that use [Azure Files](#) to store data. This article shows how to create a simple console application to do the following with .NET and Azure Files:

- Get the contents of a file.
- Set the maximum size, or quota, for a file share.
- Create a shared access signature (SAS) for a file.
- Copy a file to another file in the same storage account.
- Copy a file to a blob in the same storage account.
- Create a snapshot of a file share.
- Restore a file from a share snapshot.
- Use Azure Storage Metrics for troubleshooting.

To learn more about Azure Files, see [What is Azure Files?](#)

TIP

Check out the [Azure Storage code samples repository](#)

For easy-to-use end-to-end Azure Storage code samples that you can download and run, please check out our list of [Azure Storage Samples](#).

Applies to

FILE SHARE TYPE	SMB	NFS
Standard file shares (GPv2), LRS/ZRS	✓	✗
Standard file shares (GPv2), GRS/GZRS	✓	✗
Premium file shares (FileStorage), LRS/ZRS	✓	✗

Understanding the .NET APIs

Azure Files provides two broad approaches to client applications: Server Message Block (SMB) and REST. Within .NET, the `System.IO` and `Azure.Storage.Files.Shares` APIs abstract these approaches.

API	WHEN TO USE	NOTES
-----	-------------	-------

API	WHEN TO USE	NOTES
System.IO	Your application: <ul style="list-style-type: none"> Needs to read/write files by using SMB Is running on a device that has access over port 445 to your Azure Files account Doesn't need to manage any of the administrative settings of the file share 	File I/O implemented with Azure Files over SMB is generally the same as I/O with any network file share or local storage device. For an introduction to a number of features in .NET, including file I/O, see the Console Application tutorial .
Azure.Storage.Files.Shares	Your application: <ul style="list-style-type: none"> Can't access Azure Files by using SMB on port 445 because of firewall or ISP constraints Requires administrative functionality, such as the ability to set a file share's quota or create a shared access signature 	This article demonstrates the use of <code>Azure.Storage.Files.Shares</code> for file I/O using REST instead of SMB and management of the file share.

Create the console application and obtain the assembly

You can use the Azure Files client library in any type of .NET app. These apps include Azure cloud, web, desktop, and mobile apps. In this guide, we create a console application for simplicity.

In Visual Studio, create a new Windows console application. The following steps show you how to create a console application in Visual Studio 2019. The steps are similar in other versions of Visual Studio.

1. Start Visual Studio and select **Create a new project**.
2. In **Create a new project**, choose **Console App (.NET Framework)** for C#, and then select **Next**.
3. In **Configure your new project**, enter a name for the app, and select **Create**.

Add all the code examples in this article to the `Program` class in the *Program.cs* file.

Use NuGet to install the required packages

Refer to these packages in your project:

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)
- [Azure core library for .NET](#): This package is the implementation of the Azure client pipeline.
- [Azure Storage Blob client library for .NET](#): This package provides programmatic access to blob resources in your storage account.
- [Azure Storage Files client library for .NET](#): This package provides programmatic access to file resources in your storage account.
- [System Configuration Manager library for .NET](#): This package provides a class storing and retrieving values in a configuration file.

You can use NuGet to obtain the packages. Follow these steps:

1. In **Solution Explorer**, right-click your project and choose **Manage NuGet Packages**.

2. In **NuGet Package Manager**, select **Browse**. Then search for and choose **Azure.Core**, and then select **Install**.

This step installs the package and its dependencies.

3. Search for and install these packages:

- **Azure.Storage.Blobs**
- **Azure.Storage.Files.Shares**
- **System.Configuration.ConfigurationManager**

Save your storage account credentials to the App.config file

Next, save your credentials in your project's *App.config* file. In **Solution Explorer**, double-click `App.config` and edit the file so that it is similar to the following example.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

Replace `myaccount` with your storage account name and `mykey` with your storage account key.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="StorageConnectionString"
value="DefaultEndpointsProtocol=https;AccountName=myaccount;AccountKey=mykey;EndpointSuffix=core.windows.net" />
    <add key="StorageAccountName" value="myaccount" />
    <add key="StorageAccountKey" value="mykey" />
  </appSettings>
</configuration>
```

NOTE

The Azurite storage emulator does not currently support Azure Files. Your connection string must target an Azure storage account in the cloud to work with Azure Files.

Add using directives

In **Solution Explorer**, open the *Program.cs* file, and add the following using directives to the top of the file.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```
using System;
using System.Configuration;
using System.IO;
using System.Threading.Tasks;
using Azure;
using Azure.Storage;
using Azure.Storage.Blobs;
using Azure.Storage.Files.Shares;
using Azure.Storage.Files.Shares.Models;
using Azure.Storage.Sas;
```

Access the file share programmatically

In the *Program.cs* file, add the following code to access the file share programmatically.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

The following method creates a file share if it doesn't already exist. The method starts by creating a [ShareClient](#) object from a connection string. The sample then attempts to download a file we created earlier. Call this method from `Main()`.

```

//-----
// Create a file share
//-----
public async Task CreateShareAsync(string shareName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareClient which will be used to create and manipulate the file share
    ShareClient share = new ShareClient(connectionString, shareName);

    // Create the share if it doesn't already exist
    await share.CreateIfNotExistsAsync();

    // Ensure that the share exists
    if (await share.ExistsAsync())
    {
        Console.WriteLine($"Share created: {share.Name}");

        // Get a reference to the sample directory
        ShareDirectoryClient directory = share.GetDirectoryClient("CustomLogs");

        // Create the directory if it doesn't already exist
        await directory.CreateIfNotExistsAsync();

        // Ensure that the directory exists
        if (await directory.ExistsAsync())
        {
            // Get a reference to a file object
            ShareFileClient file = directory.GetFileClient("Log1.txt");

            // Ensure that the file exists
            if (await file.ExistsAsync())
            {
                Console.WriteLine($"File exists: {file.Name}");

                // Download the file
                ShareFileDownloadInfo download = await file.DownloadAsync();

                // Save the data to a local file, overwrite if the file already exists
                using (FileStream stream = File.OpenWrite(@"downloadedLog1.txt"))
                {
                    await download.Content.CopyToAsync(stream);
                    await stream.FlushAsync();
                    stream.Close();

                    // Display where the file was saved
                    Console.WriteLine($"File downloaded: {stream.Name}");
                }
            }
        }
    }
    else
    {
        Console.WriteLine($"CreateShareAsync failed");
    }
}

```

Set the maximum size for a file share

Beginning with version 5.x of the Azure Files client library, you can set the quota (maximum size) for a file share. You can also check to see how much data is currently stored on the share.

Setting the quota for a share limits the total size of the files stored on the share. If the total size of files on the share exceeds the quota, clients can't increase the size of existing files. Clients also can't create new files, unless

those files are empty.

The example below shows how to check the current usage for a share and how to set the quota for the share.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```
//-----  
// Set the maximum size of a share  
//-----  
public async Task SetMaxShareSizeAsync(string shareName, int increaseSizeInGiB)  
{  
    const long ONE_GIBIBYTE = 10737420000; // Number of bytes in 1 gibibyte  
  
    // Get the connection string from app settings  
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];  
  
    // Instantiate a ShareClient which will be used to access the file share  
    ShareClient share = new ShareClient(connectionString, shareName);  
  
    // Create the share if it doesn't already exist  
    await share.CreateIfNotExistsAsync();  
  
    // Ensure that the share exists  
    if (await share.ExistsAsync())  
    {  
        // Get and display current share quota  
        ShareProperties properties = await share.GetPropertiesAsync();  
        Console.WriteLine($"Current share quota: {properties.QuotaInGB} GiB");  
  
        // Get and display current usage stats for the share  
        ShareStatistics stats = await share.GetStatisticsAsync();  
        Console.WriteLine($"Current share usage: {stats.ShareUsageInBytes} bytes");  
  
        // Convert current usage from bytes into GiB  
        int currentGiB = (int)(stats.ShareUsageInBytes / ONE_GIBIBYTE);  
  
        // This line sets the quota to be the current  
        // usage of the share plus the increase amount  
        await share.SetQuotaAsync(currentGiB + increaseSizeInGiB);  
  
        // Get the new quota and display it  
        properties = await share.GetPropertiesAsync();  
        Console.WriteLine($"New share quota: {properties.QuotaInGB} GiB");  
    }  
}
```

Generate a shared access signature for a file or file share

Beginning with version 5.x of the Azure Files client library, you can generate a shared access signature (SAS) for a file share or for an individual file.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

The following example method returns a SAS on a file in the specified share.

```

//-----
// Create a SAS URI for a file
//-----
public Uri GetFileSasUri(string shareName, string filePath, DateTime expiration, ShareFileSasPermissions
permissions)
{
    // Get the account details from app settings
    string accountName = ConfigurationManager.AppSettings["StorageAccountName"];
    string accountKey = ConfigurationManager.AppSettings["StorageAccountKey"];

    ShareSasBuilder fileSAS = new ShareSasBuilder()
    {
        ShareName = shareName,
        FilePath = filePath,

        // Specify an Azure file resource
        Resource = "f",

        // Expires in 24 hours
        ExpiresOn = expiration
    };

    // Set the permissions for the SAS
    fileSAS.SetPermissions(permissions);

    // Create a SharedKeyCredential that we can use to sign the SAS token
    StorageSharedKeyCredential credential = new StorageSharedKeyCredential(accountName, accountKey);

    // Build a SAS URI
    UriBuilder fileSasUri = new
UriBuilder($"https://{accountName}.file.core.windows.net/{fileSAS.ShareName}/{fileSAS.FilePath}");
    fileSasUri.Query = fileSAS.ToSasQueryParameters(credential).ToString();

    // Return the URI
    return fileSasUri.Uri;
}

```

For more information about creating and using shared access signatures, see [How a shared access signature works](#).

Copy files

Beginning with version 5.x of the Azure Files client library, you can copy a file to another file, a file to a blob, or a blob to a file.

You can also use AzCopy to copy one file to another or to copy a blob to a file or the other way around. See [Get started with AzCopy](#).

NOTE

If you are copying a blob to a file, or a file to a blob, you must use a shared access signature (SAS) to authorize access to the source object, even if you are copying within the same storage account.

Copy a file to another file

The following example copies a file to another file in the same share. You can use [Shared Key authentication](#) to do the copy because this operation copies files within the same storage account.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```

//-----
// Copy file within a directory
//-----
public async Task CopyFileAsync(string shareName, string sourceFilePath, string destFilePath)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Get a reference to the file we created previously
    ShareFileClient sourceFile = new ShareFileClient(connectionString, shareName, sourceFilePath);

    // Ensure that the source file exists
    if (await sourceFile.ExistsAsync())
    {
        // Get a reference to the destination file
        ShareFileClient destFile = new ShareFileClient(connectionString, shareName, destFilePath);

        // Start the copy operation
        await destFile.StartCopyAsync(sourceFile.Uri);

        if (await destFile.ExistsAsync())
        {
            Console.WriteLine($"{sourceFile.Uri} copied to {destFile.Uri}");
        }
    }
}
}

```

Copy a file to a blob

The following example creates a file and copies it to a blob within the same storage account. The example creates a SAS for the source file, which the service uses to authorize access to the source file during the copy operation.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```

//-----
// Copy a file from a share to a blob
//-----
public async Task CopyFileToBlobAsync(string shareName, string sourceFilePath, string containerName, string
blobName)
{
    // Get a file SAS from the method created earlier
    Uri fileSasUri = GetFileSasUri(shareName, sourceFilePath, DateTime.UtcNow.AddHours(24),
ShareFileSasPermissions.Read);

    // Get a reference to the file we created previously
    ShareFileClient sourceFile = new ShareFileClient(fileSasUri);

    // Ensure that the source file exists
    if (await sourceFile.ExistsAsync())
    {
        // Get the connection string from app settings
        string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

        // Get a reference to the destination container
        BlobContainerClient container = new BlobContainerClient(connectionString, containerName);

        // Create the container if it doesn't already exist
        await container.CreateIfNotExistsAsync();

        BlobClient destBlob = container.GetBlobClient(blobName);

        await destBlob.StartCopyFromUriAsync(sourceFile.Uri);

        if (await destBlob.ExistsAsync())
        {
            Console.WriteLine($"File {sourceFile.Name} copied to blob {destBlob.Name}");
        }
    }
}
}

```

You can copy a blob to a file in the same way. If the source object is a blob, then create a SAS to authorize access to that blob during the copy operation.

Share snapshots

Beginning with version 8.5 of the Azure Files client library, you can create a share snapshot. You can also list or browse share snapshots and delete share snapshots. Once created, share snapshots are read-only.

Create share snapshots

The following example creates a file share snapshot.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```
//-----
// Create a share snapshot
//-----
public async Task CreateShareSnapshotAsync(string shareName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareServiceClient
    ShareServiceClient shareServiceClient = new ShareServiceClient(connectionString);

    // Instantiate a ShareClient which will be used to access the file share
    ShareClient share = shareServiceClient.GetShareClient(shareName);

    // Ensure that the share exists
    if (await share.ExistsAsync())
    {
        // Create a snapshot
        ShareSnapshotInfo snapshotInfo = await share.CreateSnapshotAsync();
        Console.WriteLine($"Snapshot created: {snapshotInfo.Snapshot}");
    }
}
}
```

List share snapshots

The following example lists the snapshots on a share.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```
//-----
// List the snapshots on a share
//-----
public void ListShareSnapshots()
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareServiceClient
    ShareServiceClient shareServiceClient = new ShareServiceClient(connectionString);

    // Display each share and the snapshots on each share
    foreach (ShareItem item in shareServiceClient.GetShares(ShareTraits.All, ShareStates.Snapshots))
    {
        if (null != item.Snapshot)
        {
            Console.WriteLine($"Share: {item.Name}\tSnapshot: {item.Snapshot}");
        }
    }
}
}
```

List files and directories within share snapshots

The following example browses files and directories within share snapshots.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)


```

//-----
// List the snapshots on a share
//-----
public void ListSnapshotContents(string shareName, string snapshotTime)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareServiceClient
    ShareServiceClient shareService = new ShareServiceClient(connectionString);

    // Get a ShareClient
    ShareClient share = shareService.GetShareClient(shareName);

    Console.WriteLine($"Share: {share.Name}");

    // Get as ShareClient that points to a snapshot
    ShareClient snapshot = share.WithSnapshot(snapshotTime);

    // Get the root directory in the snapshot share
    ShareDirectoryClient rootDir = snapshot.GetRootDirectoryClient();

    // Recursively list the directory tree
    ListDirTree(rootDir);
}

//-----
// Recursively list a directory tree
//-----
public void ListDirTree(ShareDirectoryClient dir)
{
    // List the files and directories in the snapshot
    foreach (ShareFileItem item in dir.GetFilesAndDirectories())
    {
        if (item.IsDirectory)
        {
            Console.WriteLine($"Directory: {item.Name}");
            ShareDirectoryClient subDir = dir.GetSubdirectoryClient(item.Name);
            ListDirTree(subDir);
        }
        else
        {
            Console.WriteLine($"File: {dir.Name}\\{item.Name}");
        }
    }
}
}

```

Restore file shares or files from share snapshots

Taking a snapshot of a file share enables you to recover individual files or the entire file share.

You can restore a file from a file share snapshot by querying the share snapshots of a file share. You can then retrieve a file that belongs to a particular share snapshot. Use that version to directly read or to restore the file.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```

//-----
// Restore file from snapshot
//-----
public async Task RestoreFileFromSnapshot(string shareName, string directoryName, string fileName, string
snapshotTime)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareServiceClient
    ShareServiceClient shareService = new ShareServiceClient(connectionString);

    // Get a ShareClient
    ShareClient share = shareService.GetShareClient(shareName);

    // Get as ShareClient that points to a snapshot
    ShareClient snapshot = share.WithSnapshot(snapshotTime);

    // Get a ShareDirectoryClient, then a ShareFileClient to the snapshot file
    ShareDirectoryClient snapshotDir = snapshot.GetDirectoryClient(directoryName);
    ShareFileClient snapshotFile = snapshotDir.GetFileClient(fileName);

    // Get a ShareDirectoryClient, then a ShareFileClient to the live file
    ShareDirectoryClient liveDir = share.GetDirectoryClient(directoryName);
    ShareFileClient liveFile = liveDir.GetFileClient(fileName);

    // Restore the file from the snapshot
    ShareFileCopyInfo copyInfo = await liveFile.StartCopyAsync(snapshotFile.Uri);

    // Display the status of the operation
    Console.WriteLine($"Restore status: {copyInfo.CopyStatus}");
}

```

Delete share snapshots

The following example deletes a file share snapshot.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```

//-----
// Delete a snapshot
//-----
public async Task DeleteSnapshotAsync(string shareName, string snapshotTime)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareServiceClient
    ShareServiceClient shareService = new ShareServiceClient(connectionString);

    // Get a ShareClient
    ShareClient share = shareService.GetShareClient(shareName);

    // Get a ShareClient that points to a snapshot
    ShareClient snapshotShare = share.WithSnapshot(snapshotTime);

    try
    {
        // Delete the snapshot
        await snapshotShare.DeleteIfExistsAsync();
    }
    catch (RequestFailedException ex)
    {
        Console.WriteLine($"Exception: {ex.Message}");
        Console.WriteLine($"Error code: {ex.Status}\t{ex.ErrorCode}");
    }
}

```

Troubleshoot Azure Files by using metrics

Azure Storage Analytics supports metrics for Azure Files. With metrics data, you can trace requests and diagnose issues.

You can enable metrics for Azure Files from the [Azure portal](#). You can also enable metrics programmatically by calling the [Set File Service Properties](#) operation with the REST API or one of its analogs in the Azure Files client library.

The following code example shows how to use the .NET client library to enable metrics for Azure Files.

- [Azure .NET SDK v12](#)
- [Azure .NET SDK v11](#)

```

//-----
// Use metrics
//-----
public async Task UseMetricsAsync()
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a ShareServiceClient
    ShareServiceClient shareService = new ShareServiceClient(connectionString);

    // Set metrics properties for File service
    await shareService.SetPropertiesAsync(new ShareServiceProperties()
    {
        // Set hour metrics
        HourMetrics = new ShareMetrics()
        {
            Enabled = true,
            IncludeApis = true,
            Version = "1.0",

            RetentionPolicy = new ShareRetentionPolicy()
            {
                Enabled = true,
                Days = 14
            }
        },

        // Set minute metrics
        MinuteMetrics = new ShareMetrics()
        {
            Enabled = true,
            IncludeApis = true,
            Version = "1.0",

            RetentionPolicy = new ShareRetentionPolicy()
            {
                Enabled = true,
                Days = 7
            }
        }
    });

    // Read the metrics properties we just set
    ShareServiceProperties serviceProperties = await shareService.GetPropertiesAsync();

    // Display the properties
    Console.WriteLine();
    Console.WriteLine($"HourMetrics.IncludeApis: {serviceProperties.HourMetrics.IncludeApis}");
    Console.WriteLine($"HourMetrics.RetentionPolicy.Days:
{serviceProperties.HourMetrics.RetentionPolicy.Days}");
    Console.WriteLine($"HourMetrics.Version: {serviceProperties.HourMetrics.Version}");
    Console.WriteLine();
    Console.WriteLine($"MinuteMetrics.IncludeApis: {serviceProperties.MinuteMetrics.IncludeApis}");
    Console.WriteLine($"MinuteMetrics.RetentionPolicy.Days:
{serviceProperties.MinuteMetrics.RetentionPolicy.Days}");
    Console.WriteLine($"MinuteMetrics.Version: {serviceProperties.MinuteMetrics.Version}");
    Console.WriteLine();
}

```

If you encounter any problems, you can refer to [Troubleshoot Azure Files problems in Windows](#).

Next steps

For more information about Azure Files, see the following resources:

Conceptual articles and videos

- [Azure Files: a frictionless cloud SMB file system for Windows and Linux](#)
- [Use Azure Files with Linux](#)

Tooling support for File storage

- [Get started with AzCopy](#)
- [Troubleshoot Azure Files problems in Windows](#)

Reference

- [Azure Storage APIs for .NET](#)
- [File Service REST API](#)

Quickstart: Build a Table API app with .NET SDK and Azure Cosmos DB

6/16/2021 • 9 minutes to read • [Edit Online](#)

APPLIES TO:  Table API

This quickstart shows how to use .NET and the Azure Cosmos DB [Table API](#) to build an app by cloning an example from GitHub. This quickstart also shows you how to create an Azure Cosmos DB account and how to use Data Explorer to create tables and entities in the web-based Azure portal.

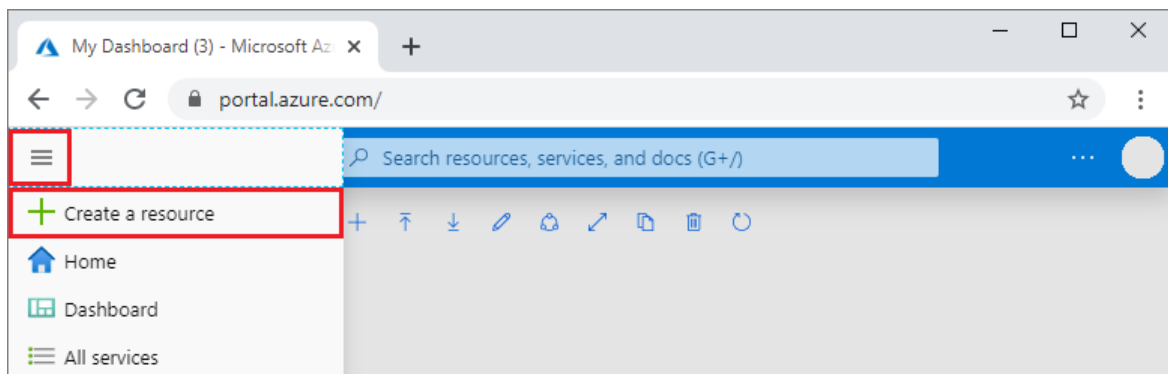
Prerequisites

If you don't already have Visual Studio 2019 installed, you can download and use the **free** [Visual Studio 2019 Community Edition](#). Make sure that you enable **Azure development** during the Visual Studio setup.

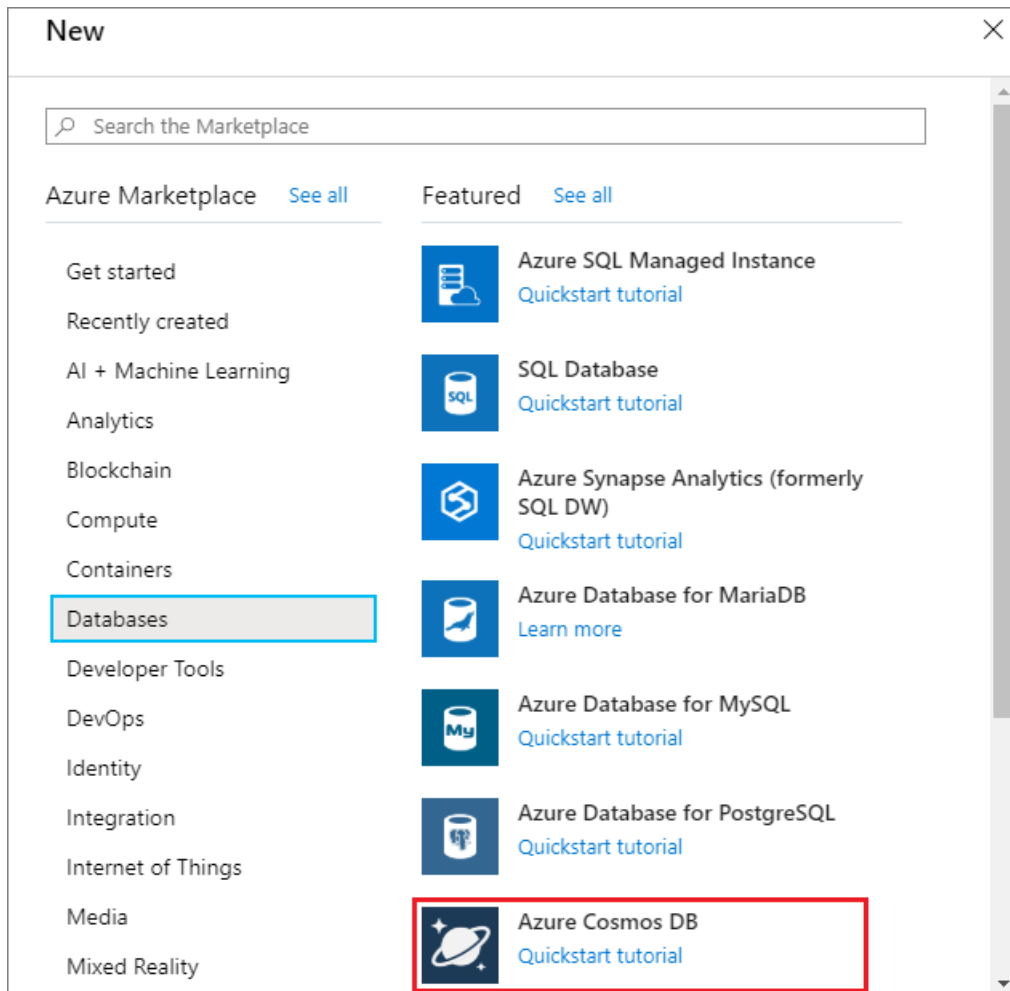
If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Create a database account

1. In a new browser window, sign in to the [Azure portal](#).
2. In the left menu, select **Create a resource**.



3. On the **New** page, select **Databases > Azure Cosmos DB**.



4. On the **Create Azure Cosmos DB Account** page, enter the settings for the new Azure Cosmos DB account.

SETTING	VALUE	DESCRIPTION
Subscription	Your subscription	Select the Azure subscription that you want to use for this Azure Cosmos DB account.
Resource Group	Create new , then Account Name	Select Create new . Then enter a new resource group name for your account. For simplicity, use the same name as your Azure Cosmos DB account name.
Account Name	A unique name	Enter a unique name to identify your Azure Cosmos DB account. The account name can use only lowercase letters, numbers, and hyphens (-), and must be between 3 and 31 characters long.

SETTING	VALUE	DESCRIPTION
API	Table	<p>The API determines the type of account to create. Azure Cosmos DB provides five APIs: Core (SQL) for document databases, Gremlin for graph databases, MongoDB for document databases, Azure Table, and Cassandra. You must create a separate account for each API.</p> <p>Select Azure Table, because in this quickstart you are creating a table that works with the Table API.</p> <p>Learn more about the Table API.</p>
Location	The region closest to your users	<p>Select a geographic location to host your Azure Cosmos DB account. Use the location that's closest to your users to give them the fastest access to the data.</p>
Capacity mode	Provisioned throughput or Serverless	<p>Select Provisioned throughput to create an account in provisioned throughput mode. Select Serverless to create an account in serverless mode.</p>

You can leave the **Geo-Redundancy** and **Multi-region Writes** options at **Disable** to avoid additional charges, and skip the **Network** and **Tags** sections.

5. Select **Review + Create**. After the validation is complete, select **Create** to create the account.

Home > New >

Create Azure Cosmos DB Account

Basics Networking Backup Policy Encryption Tags Review + create

Azure Cosmos DB is a globally distributed, multi-model, fully managed database service. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

Project Details

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

Subscription *

Resource Group *

[Create new](#)

Instance Details

Account Name *

API *

Notebooks (Preview) On Off

Location *

Capacity mode Provisioned throughput Serverless (preview)

[Learn more about capacity mode](#)

With Azure Cosmos DB free tier, you will get 400 RU/s and 5 GB of storage for free in an account. You can enable free tier on up to one account per subscription. Estimated \$24/month discount per account.

Apply Free Tier Discount Apply Do Not Apply

Account Type Production Non-Production

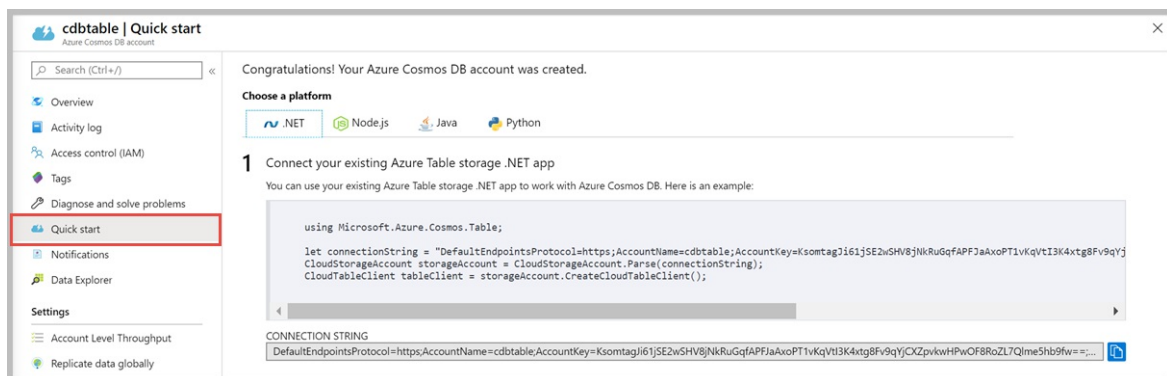
Geo-Redundancy Enable Disable

Multi-region Writes Enable Disable

*Up to 33% off multi-region writes is available to qualifying new accounts only. Offer limited to accounts with both account locations and geo-redundancy, and applies only to multi-region writes in those same regions. Both Geo-Redundancy and Multi-region Writes must be enabled in account settings. Actual discount will vary based on number of qualifying regions selected.

[Review + create](#) [Previous](#) [Next: Networking](#)

6. It takes a few minutes to create the account. You'll see a message that states **Your deployment is underway**. Wait for the deployment to finish, and then select **Go to resource**.



cdbtable | Quick start

Azure Cosmos DB account

Search (Ctrl+F)

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Quick start
Notifications
Data Explorer
Settings
Account Level Throughput
Replicate data globally

Congratulations! Your Azure Cosmos DB account was created.

Choose a platform

.NET Node.js Java Python

1 Connect your existing Azure Table storage .NET app

You can use your existing Azure Table storage .NET app to work with Azure Cosmos DB. Here is an example:

```
using Microsoft.Azure.Cosmos.Table;

let connectionString = "DefaultEndpointsProtocol=https;AccountName=cdbtable;AccountKey=KsontagJi61jSE2wSHV8jNkRuGqAPF3aAoxPT1vKqVtI3K4xtg8Fv9qj;CloudStorageAccount=storageAccount=CloudStorageAccount.Parse(connectionString);CloudTableClient tableClient = storageAccount.CreateCloudTableClient();
```

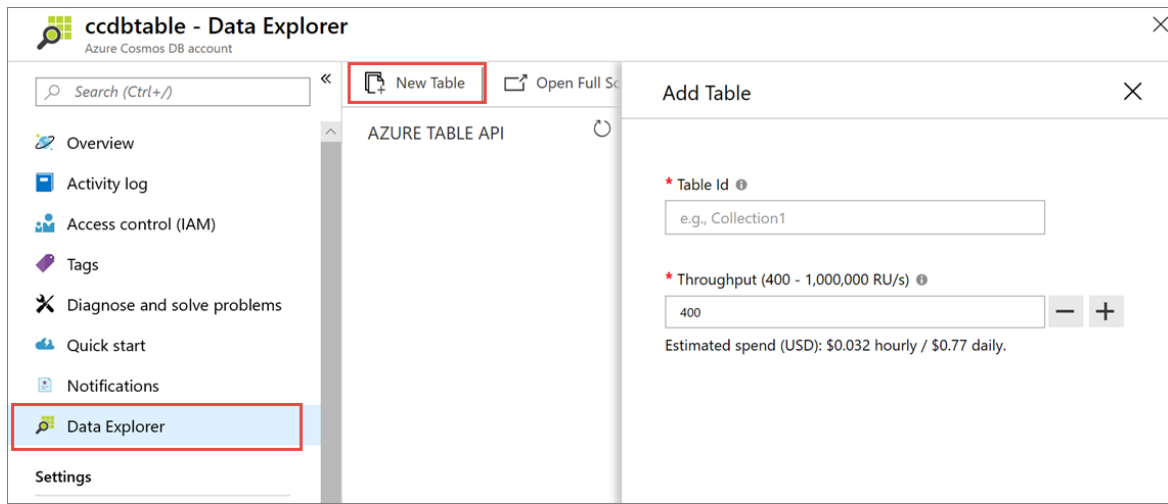
CONNECTION STRING
DefaultEndpointsProtocol=https;AccountName=cdbtable;AccountKey=KsontagJi61jSE2wSHV8jNkRuGqAPF3aAoxPT1vKqVtI3K4xtg8Fv9qj;CXZpvkWhPwOF8RoZL7Qlme5hb9fw=...

Add a table

You can now use the Data Explorer tool in the Azure portal to create a database and table.

1. Select **Data Explorer > New Table**.

The **Add Table** area is displayed on the far right, you may need to scroll right to see it.

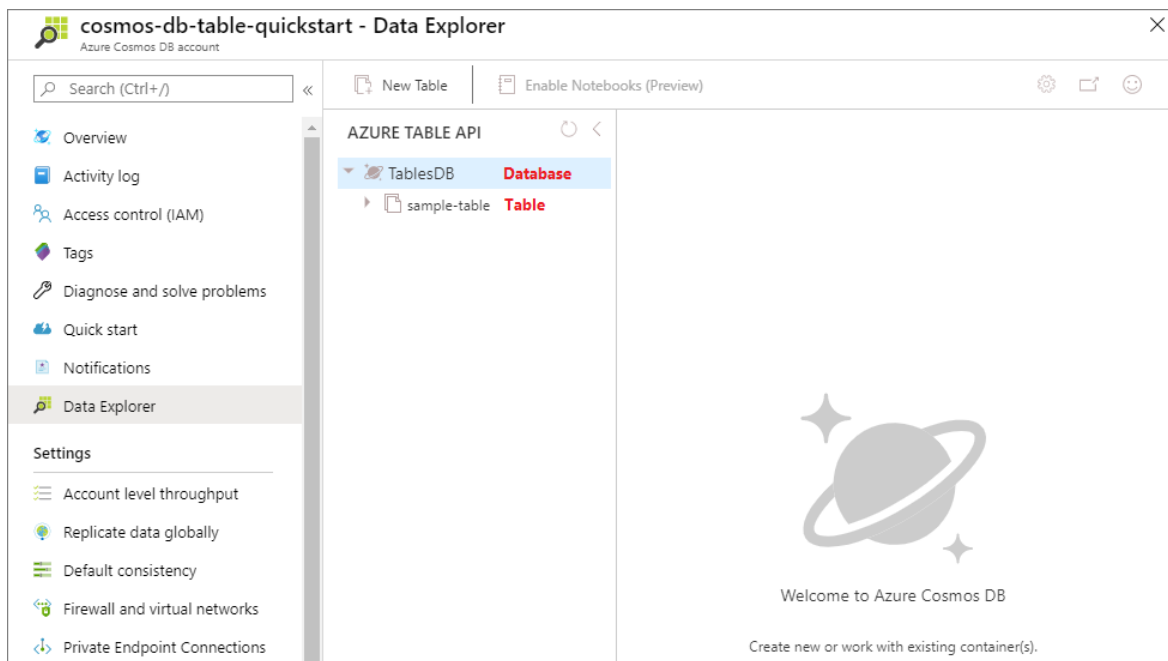


2. In the Add Table page, enter the settings for the new table.

SETTING	SUGGESTED VALUE	DESCRIPTION
Table Id	sample-table	The ID for your new table. Table names have the same character requirements as database ids. Database names must be between 1 and 255 characters, and cannot contain <code>/ \ # ?</code> or a trailing space.
Throughput	400 RUs	Change the throughput to 400 request units per second (RU/s). If you want to reduce latency, you can scale up the throughput later.

3. Select OK.

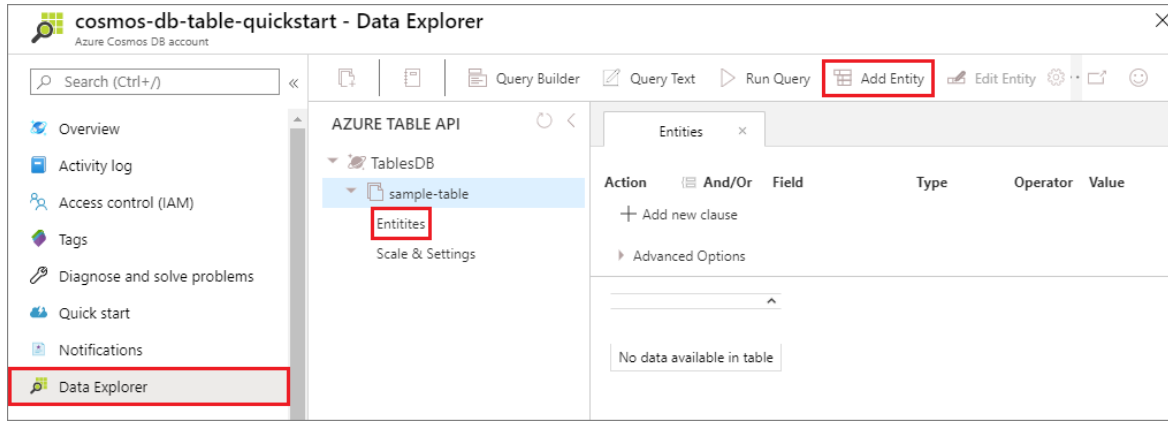
4. Data Explorer displays the new database and table.



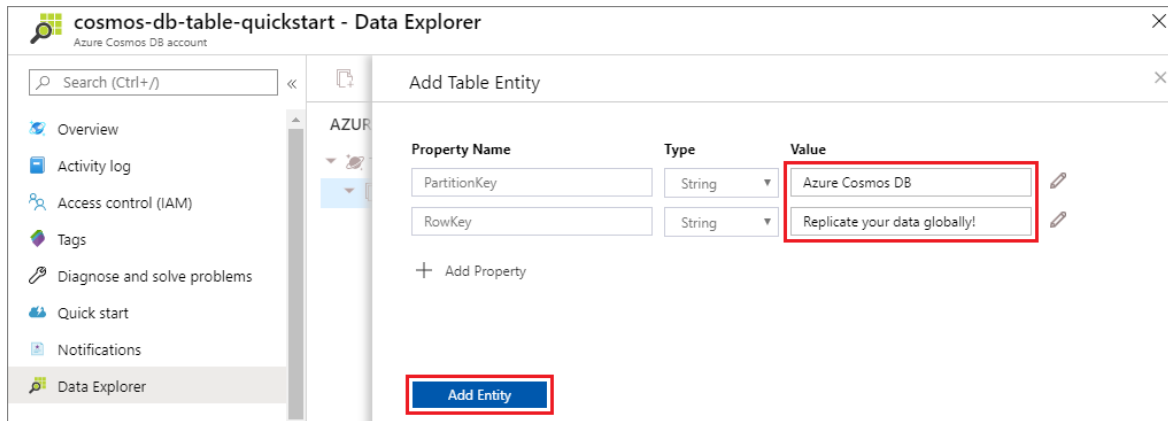
Add sample data

You can now add data to your new table using Data Explorer.

1. In Data Explorer, expand **sample-table**, select **Entities**, and then select **Add Entity**.



2. Now add data to the PartitionKey value box and RowKey value box, and select **Add Entity**.



You can now add more entities to your table, edit your entities, or query your data in Data Explorer. Data Explorer is also where you can scale your throughput and add stored procedures, user-defined functions, and triggers to your table.

Clone the sample application

Now let's clone a Table app from GitHub, set the connection string, and run it. You'll see how easy it is to work with data programmatically.

1. Open a command prompt, create a new folder named `git-samples`, then close the command prompt.

```
md "C:\git-samples"
```

2. Open a git terminal window, such as `git bash`, and use the `cd` command to change to the new folder to install the sample app.

```
cd "C:\git-samples"
```

3. Run the following command to clone the sample repository. This command creates a copy of the sample app on your computer.

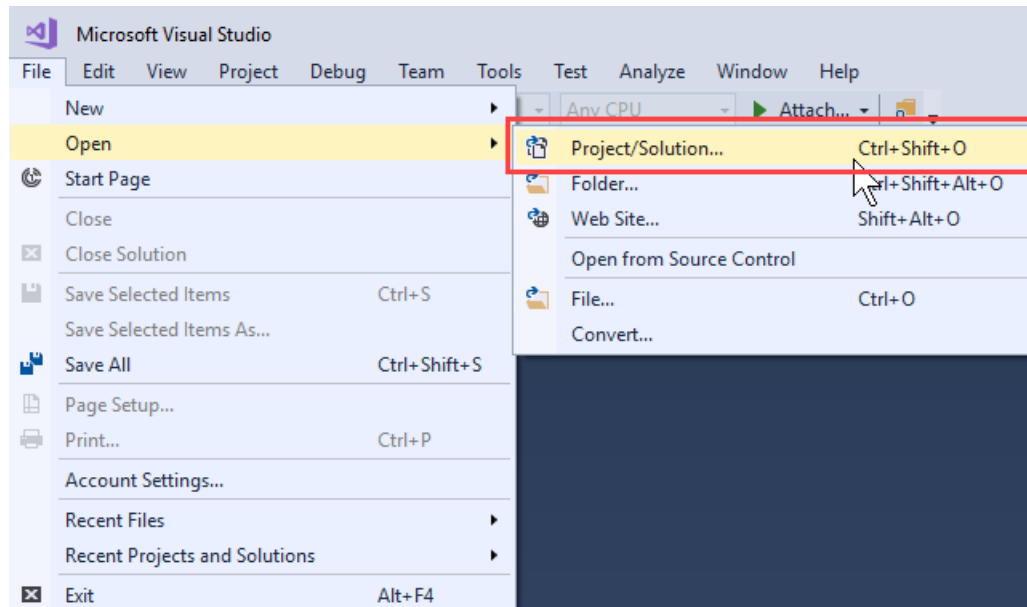
```
git clone https://github.com/Azure-Samples/azure-cosmos-table-dotnet-core-getting-started.git
```

TIP

For a more detailed walkthrough of similar code, see the [Cosmos DB Table API sample](#) article.

Open the sample application in Visual Studio

1. In Visual Studio, from the **File** menu, choose **Open**, then choose **Project/Solution**.



2. Navigate to the folder where you cloned the sample application and open the TableStorage.sln file.

Review the code

This step is optional. If you're interested in learning how the database resources are created in the code, you can review the following snippets. Otherwise, you can skip ahead to [update the connection string](#) section of this doc.

- The following code shows how to create a table within the Azure Storage:

```

public static async Task<CloudTable> CreateTableAsync(string tableName)
{
    string storageConnectionString = AppSettings.LoadAppSettings().StorageConnectionString;

    // Retrieve storage account information from connection string.
    CloudStorageAccount storageAccount =
    CreateStorageAccountFromConnectionString(storageConnectionString);

    // Create a table client for interacting with the table service
    CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new
    TableClientConfiguration());

    Console.WriteLine("Create a Table for the demo");

    // Create a table client for interacting with the table service
    CloudTable table = tableClient.GetTableReference(tableName);
    if (await table.CreateIfNotExistsAsync())
    {
        Console.WriteLine("Created Table named: {0}", tableName);
    }
    else
    {
        Console.WriteLine("Table {0} already exists", tableName);
    }

    Console.WriteLine();
    return table;
}

```

- The following code shows how to insert data into the table:

```

public static async Task<CustomerEntity> InsertOrMergeEntityAsync(CloudTable table, CustomerEntity
entity)
{
    if (entity == null)
    {
        throw new ArgumentNullException("entity");
    }

    try
    {
        // Create the InsertOrReplace table operation
        TableOperation insertOrMergeOperation = TableOperation.InsertOrMerge(entity);

        // Execute the operation.
        TableResult result = await table.ExecuteAsync(insertOrMergeOperation);
        CustomerEntity insertedCustomer = result.Result as CustomerEntity;

        if (result.RequestCharge.HasValue)
        {
            Console.WriteLine("Request Charge of InsertOrMerge Operation: " + result.RequestCharge);
        }

        return insertedCustomer;
    }
    catch (StorageException e)
    {
        Console.WriteLine(e.Message);
        Console.ReadLine();
        throw;
    }
}

```

- The following code shows how to query data from the table:

```

public static async Task<CustomerEntity> RetrieveEntityUsingPointQueryAsync(CloudTable table, string
partitionKey, string rowKey)
{
    try
    {
        TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>(partitionKey,
rowKey);
        TableResult result = await table.ExecuteAsync(retrieveOperation);
        CustomerEntity customer = result.Result as CustomerEntity;
        if (customer != null)
        {
            Console.WriteLine("\t{0}\t{1}\t{2}\t{3}", customer.PartitionKey, customer.RowKey,
customer.Email, customer.PhoneNumber);
        }

        if (result.RequestCharge.HasValue)
        {
            Console.WriteLine("Request Charge of Retrieve Operation: " + result.RequestCharge);
        }

        return customer;
    }
    catch (StorageException e)
    {
        Console.WriteLine(e.Message);
        Console.ReadLine();
        throw;
    }
}

```

- The following code shows how to delete data from the table:

```

public static async Task DeleteEntityAsync(CloudTable table, CustomerEntity deleteEntity)
{
    try
    {
        if (deleteEntity == null)
        {
            throw new ArgumentNullException("deleteEntity");
        }

        TableOperation deleteOperation = TableOperation.Delete(deleteEntity);
        TableResult result = await table.ExecuteAsync(deleteOperation);

        if (result.RequestCharge.HasValue)
        {
            Console.WriteLine("Request Charge of Delete Operation: " + result.RequestCharge);
        }

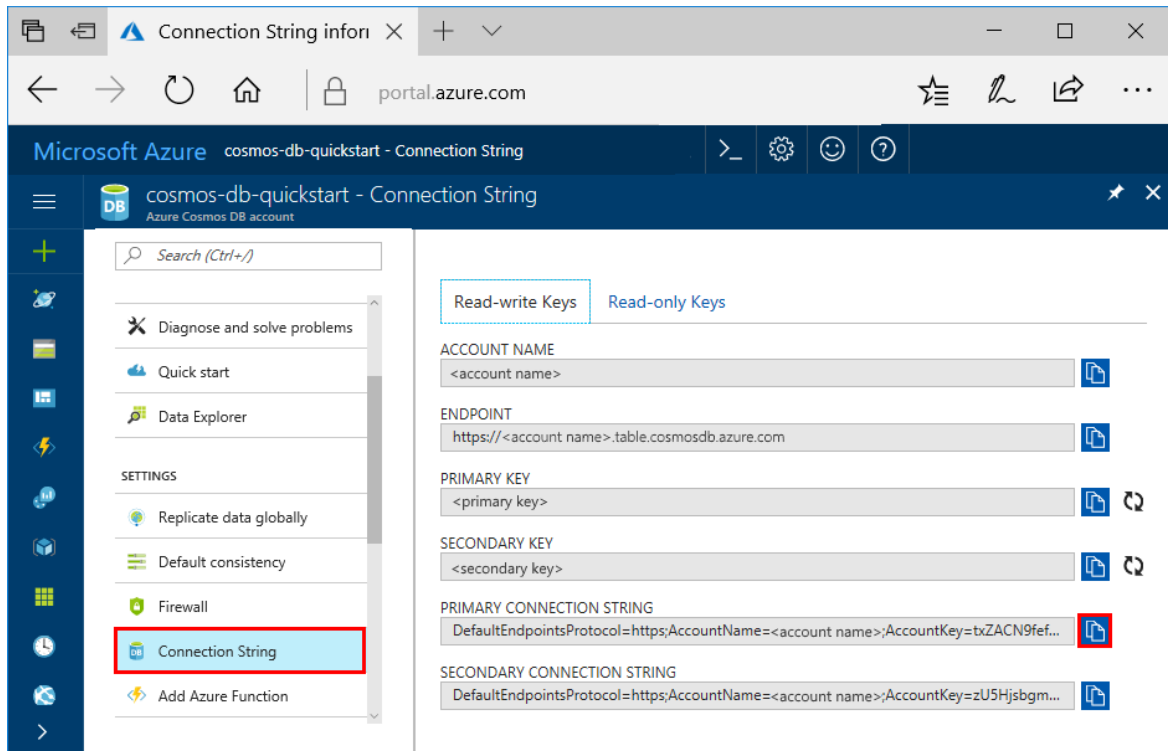
    }
    catch (StorageException e)
    {
        Console.WriteLine(e.Message);
        Console.ReadLine();
        throw;
    }
}

```

Update your connection string

Now go back to the Azure portal to get your connection string information and copy it into the app. This enables your app to communicate with your hosted database.

1. In the [Azure portal](#), click **Connection String**. Use the copy button on the right side of the window to copy the **PRIMARY CONNECTION STRING**.



2. In Visual Studio, open the **Settings.json** file.
3. Paste the **PRIMARY CONNECTION STRING** from the portal into the **StorageConnectionString** value. Paste the string inside the quotes.

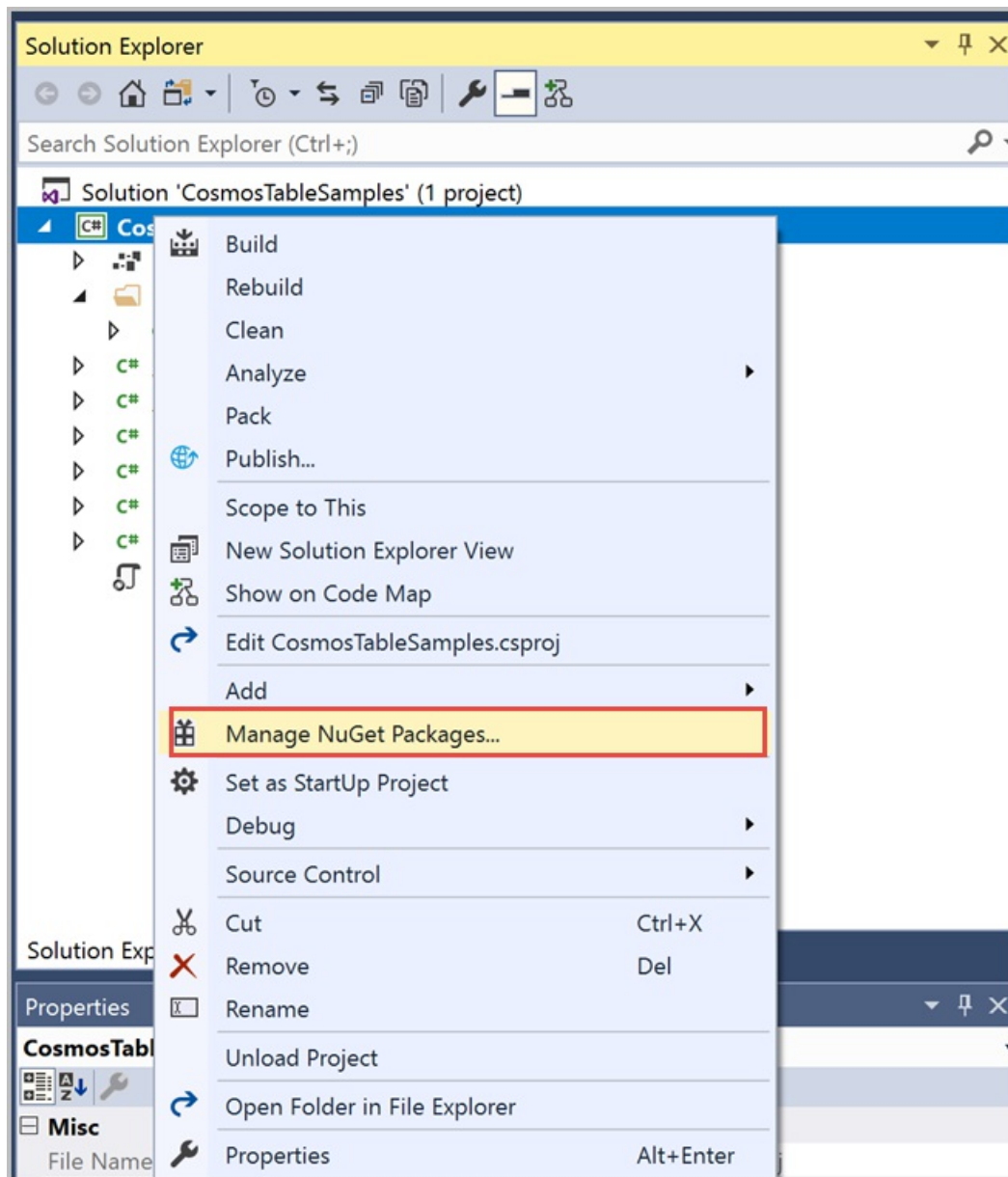
```
{
  "StorageConnectionString": "<Primary connection string from Azure portal>"
}
```

4. Press CTRL+S to save the **Settings.json** file.

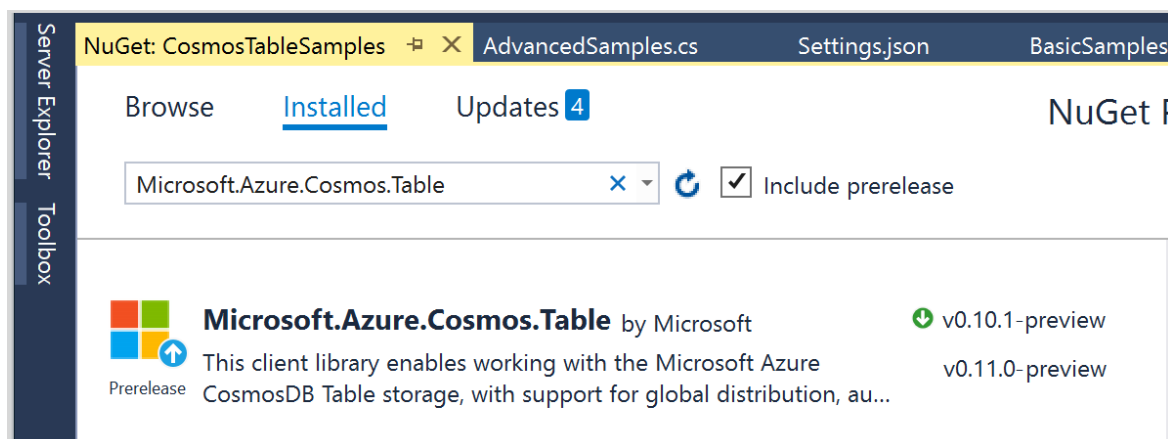
You've now updated your app with all the info it needs to communicate with Azure Cosmos DB.

Build and deploy the app

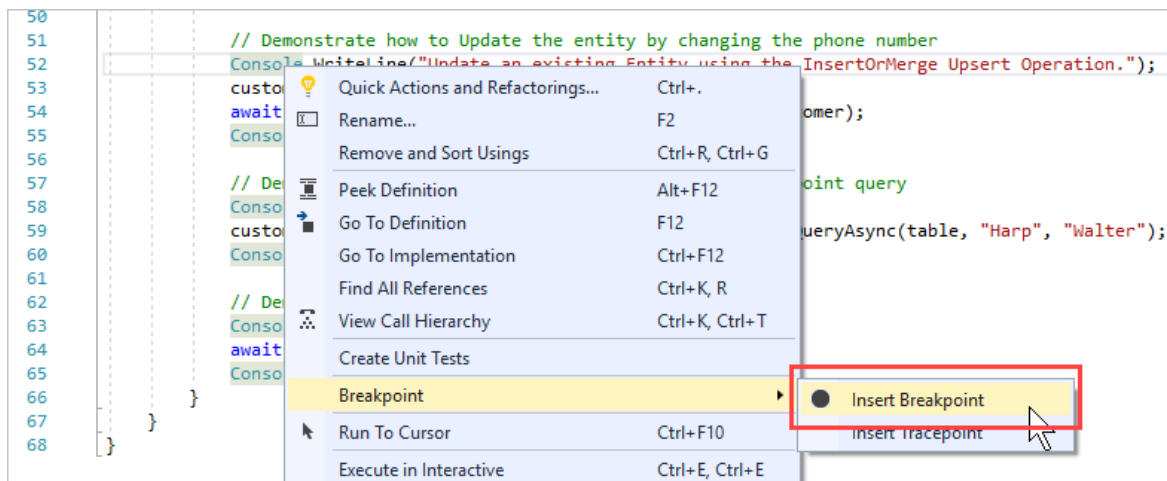
1. In Visual Studio, right-click on the **CosmosTableSamples** project in **Solution Explorer** and then click **Manage NuGet Packages**.



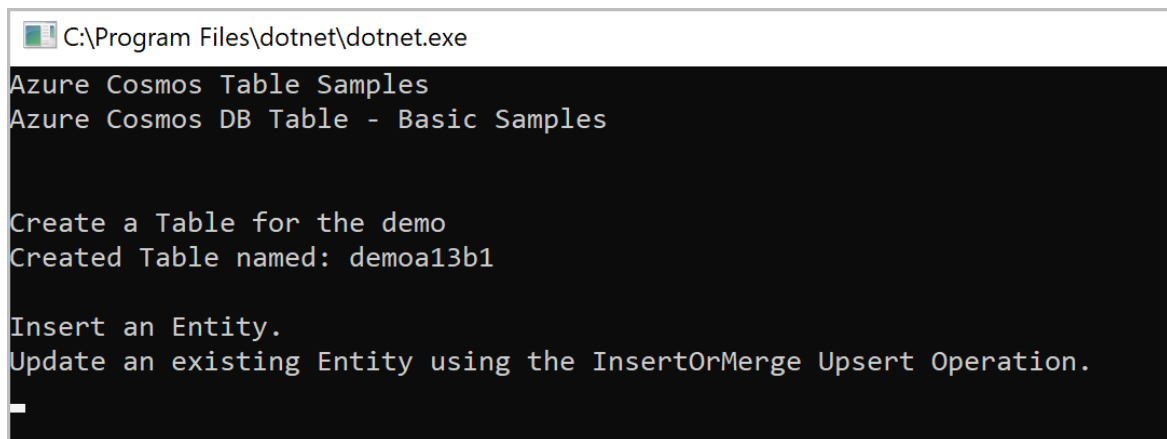
2. In the NuGet **Browse** box, type Microsoft.Azure.Cosmos.Table. This will find the Cosmos DB Table API client library. Note that this library is currently available for .NET Framework and .NET Standard.



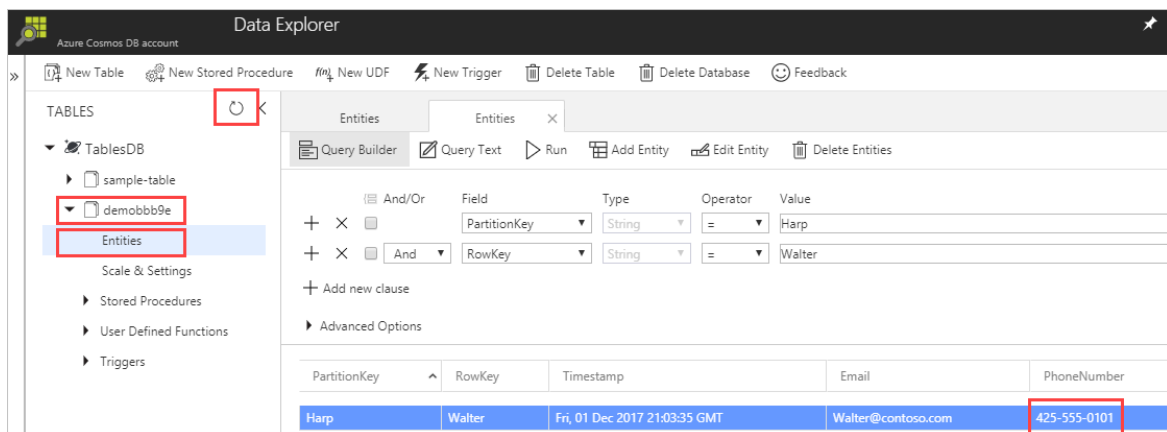
3. Click **Install** to install the **Microsoft.Azure.Cosmos.Table** library. This installs the Azure Cosmos DB Table API package and all dependencies.
4. When you run the entire app, sample data is inserted into the table entity and deleted at the end so you won't see any data inserted if you run the whole sample. However you can insert some breakpoints to view the data. Open BasicSamples.cs file and right-click on line 52, select **Breakpoint**, then select **Insert Breakpoint**. Insert another breakpoint on line 55.



- Press F5 to run the application. The console window displays the name of the new table database (in this case, demoa13b1) in Azure Cosmos DB.



When you hit the first breakpoint, go back to Data Explorer in the Azure portal. Click the **Refresh** button, expand the demo* table, and click **Entities**. The **Entities** tab on the right shows the new entity that was added for Walter Harp. Note that the phone number for the new entity is 425-555-0101.



If you receive an error that says Settings.json file can't be found when running the project, you can resolve it by adding the following XML entry to the project settings. Right click on CosmosTableSamples, select Edit CosmosTableSamples.csproj and add the following itemGroup:

```

<ItemGroup>
  <None Update="Settings.json">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>

```

6. Close the **Entities** tab in Data Explorer.
7. Press F5 to run the app to the next breakpoint.

When you hit the breakpoint, switch back to the Azure portal, click **Entities** again to open the **Entities** tab, and note that the phone number has been updated to 425-555-0105.

8. Press F5 to run the app.

The app adds entities for use in an advanced sample app that the Table API currently does not support. The app then deletes the table created by the sample app.

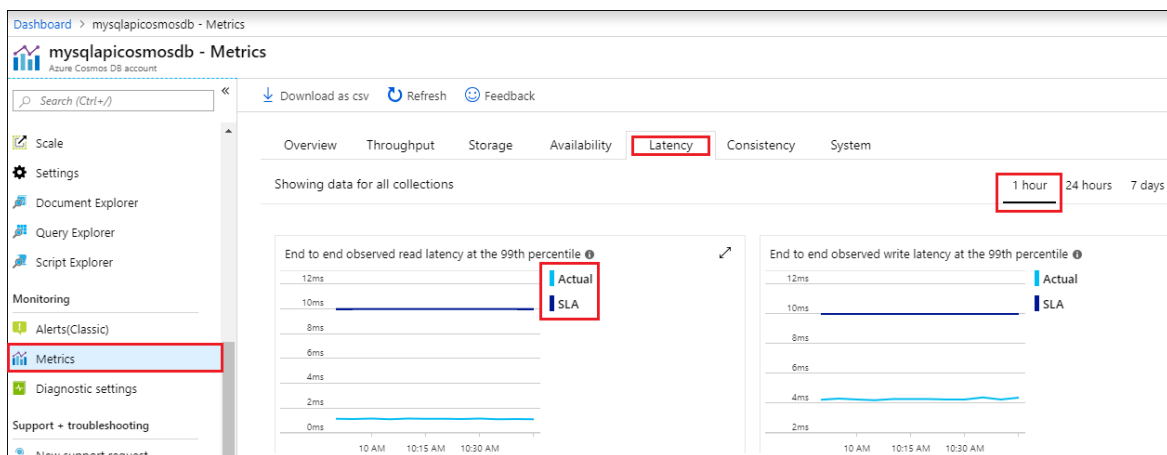
9. In the console window, press Enter to end the execution of the app.

Review SLAs in the Azure portal

The Azure portal monitors your Cosmos DB account throughput, storage, availability, latency, and consistency. Charts for metrics associated with an [Azure Cosmos DB Service Level Agreement \(SLA\)](#) show the SLA value compared to actual performance. This suite of metrics makes monitoring your SLAs transparent.

To review metrics and SLAs:

1. Select **Metrics** in your Cosmos DB account's navigation menu.
2. Select a tab such as **Latency**, and select a timeframe on the right. Compare the **Actual** and **SLA** lines on the charts.

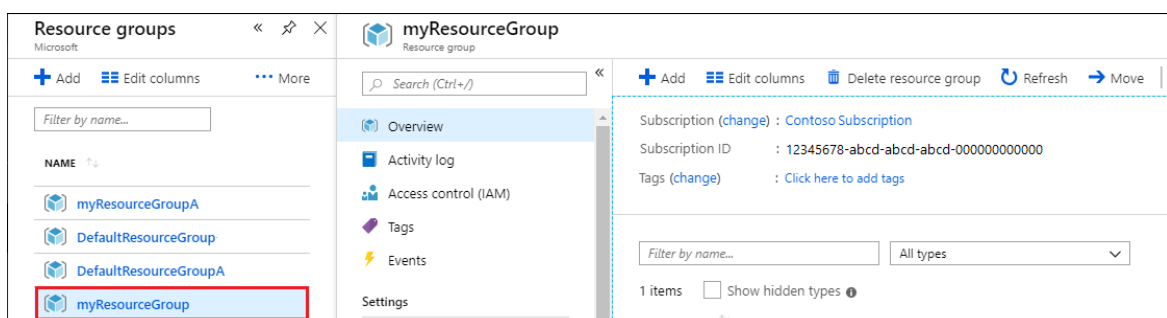


3. Review the metrics on the other tabs.

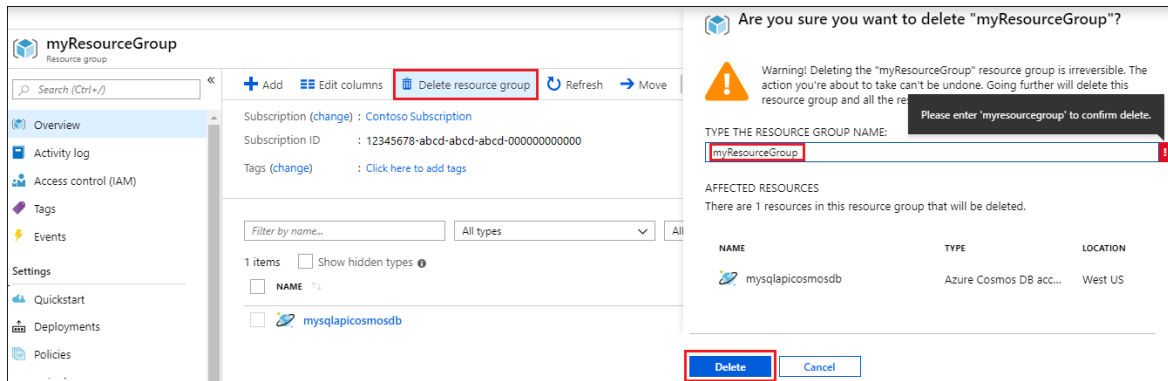
Clean up resources

When you're done with your app and Azure Cosmos DB account, you can delete the Azure resources you created so you don't incur more charges. To delete the resources:

1. In the Azure portal Search bar, search for and select **Resource groups**.
2. From the list, select the resource group you created for this quickstart.



3. On the resource group **Overview** page, select **Delete resource group**.



4. In the next window, enter the name of the resource group to delete, and then select **Delete**.


Next steps

In this quickstart, you've learned how to create an Azure Cosmos DB account, create a table using the Data Explorer, and run an app. Now you can query your data using the Table API.

[Import table data to the Table API](#)

Quickstart: Build a .NET console app to manage Azure Cosmos DB SQL API resources

4/26/2021 • 12 minutes to read • [Edit Online](#)

APPLIES TO:  SQL API

Get started with the Azure Cosmos DB SQL API client library for .NET. Follow the steps in this doc to install the .NET package, build an app, and try out the example code for basic CRUD operations on the data stored in Azure Cosmos DB.

Azure Cosmos DB is Microsoft's fast NoSQL database with open APIs for any scale. You can use Azure Cosmos DB to quickly create and query key/value, document, and graph databases. Use the Azure Cosmos DB SQL API client library for .NET to:

- Create an Azure Cosmos database and a container
- Add sample data to the container
- Query the data
- Delete the database

[API reference documentation](#) | [Library source code](#) | [Package \(NuGet\)](#)

Prerequisites

- Azure subscription - [create one for free](#) or you can [Try Azure Cosmos DB for free](#) without an Azure subscription, free of charge and commitments.
- The [.NET Core 2.1 SDK or later](#).

Setting up

This section walks you through creating an Azure Cosmos account and setting up a project that uses Azure Cosmos DB SQL API client library for .NET to manage resources. The example code described in this article creates a `FamilyDatabase` database and family members (each family member is an item) within that database. Each family member has properties such as `Id, FamilyName, FirstName, LastName, Parents, Children, Address,`. The `LastName` property is used as the partition key for the container.

Create an Azure Cosmos account

If you use the [Try Azure Cosmos DB for free](#) option to create an Azure Cosmos account, you must create an Azure Cosmos DB account of type **SQL API**. An Azure Cosmos DB test account is already created for you. You don't have to create the account explicitly, so you can skip this section and move to the next section.

If you have your own Azure subscription or created a subscription for free, you should create an Azure Cosmos account explicitly. The following code will create an Azure Cosmos account with session consistency. The account is replicated in `South Central US` and `North Central US`.

You can use Azure Cloud Shell to create the Azure Cosmos account. Azure Cloud Shell is an interactive, authenticated, browser-accessible shell for managing Azure resources. It provides the flexibility of choosing the shell experience that best suits the way you work, either Bash or PowerShell. For this quickstart, choose **Bash** mode. Azure Cloud Shell also requires a storage account, you can create one when prompted.

Select the **Try It** button next to the following code, choose **Bash** mode select **create a storage account** and login to Cloud Shell. Next copy and paste the following code to Azure Cloud Shell and run it. The Azure Cosmos

account name must be globally unique, make sure to update the `mysqlapicosmosdb` value before you run the command.

```
# Set variables for the new SQL API account, database, and container
resourceGroupName='myResourceGroup'
location='southcentralus'

# The Azure Cosmos account name must be globally unique, make sure to update the `mysqlapicosmosdb` value
before you run the command
accountName='mysqlapicosmosdb'

# Create a resource group
az group create \
  --name $resourceGroupName \
  --location $location

# Create a SQL API Cosmos DB account with session consistency and multi-region writes enabled
az cosmosdb create \
  --resource-group $resourceGroupName \
  --name $accountName \
  --kind GlobalDocumentDB \
  --locations regionName="South Central US" failoverPriority=0 --locations regionName="North Central US"
failoverPriority=1 \
  --default-consistency-level "Session" \
  --enable-multiple-write-locations true
```

The creation of the Azure Cosmos account takes a while, once the operation is successful, you can see the confirmation output. After the command completes successfully, sign into the [Azure portal](#) and verify that the Azure Cosmos account with the specified name exists. You can close the Azure Cloud Shell window after the resource is created.

Create a new .NET app

Create a new .NET application in your preferred editor or IDE. Open the Windows command prompt or a Terminal window from your local computer. You will run all the commands in the next sections from the command prompt or terminal. Run the following `dotnet new` command to create a new app with the name `todo`. The `--langVersion` parameter sets the `LangVersion` property in the created project file.

```
dotnet new console --langVersion 7.1 -n todo
```

Change your directory to the newly created app folder. You can build the application with:

```
cd todo
dotnet build
```

The expected output from the build should look something like this:

```
Restore completed in 100.37 ms for C:\Users\user1\Downloads\CosmosDB_Samples\todo\todo.csproj.
todo -> C:\Users\user1\Downloads\CosmosDB_Samples\todo\bin\Debug\netcoreapp2.2\todo.dll
todo -> C:\Users\user1\Downloads\CosmosDB_Samples\todo\bin\Debug\netcoreapp2.2\todo.Views.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:34.17
```

Install the Azure Cosmos DB package

While still in the application directory, install the Azure Cosmos DB client library for .NET Core by using the `dotnet add package` command.

```
dotnet add package Microsoft.Azure.Cosmos
```

Copy your Azure Cosmos account credentials from the Azure portal

The sample application needs to authenticate to your Azure Cosmos account. To authenticate, you should pass the Azure Cosmos account credentials to the application. Get your Azure Cosmos account credentials by following these steps:

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Cosmos account.
3. Open the **Keys** pane and copy the **URI** and **PRIMARY KEY** of your account. You will add the URI and keys values to an environment variable in the next step.

Set the environment variables

After you have copied the **URI** and **PRIMARY KEY** of your account, save them to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and run the following command. Make sure to replace `<Your_Azure_Cosmos_account_URI>` and `<Your_Azure_Cosmos_account_PRIMARY_KEY>` values.

Windows

```
setx EndpointUrl "<Your_Azure_Cosmos_account_URI>"
setx PrimaryKey "<Your_Azure_Cosmos_account_PRIMARY_KEY>"
```

Linux

```
export EndpointUrl = "<Your_Azure_Cosmos_account_URI>"
export PrimaryKey = "<Your_Azure_Cosmos_account_PRIMARY_KEY>"
```

macOS

```
export EndpointUrl = "<Your_Azure_Cosmos_account_URI>"
export PrimaryKey = "<Your_Azure_Cosmos_account_PRIMARY_KEY>"
```

Object model

Before you start building the application, let's look into the hierarchy of resources in Azure Cosmos DB and the object model used to create and access these resources. The Azure Cosmos DB creates resources in the following order:

- Azure Cosmos account
- Databases
- Containers
- Items

To learn in more about the hierarchy of different entities, see the [working with databases, containers, and items in Azure Cosmos DB](#) article. You will use the following .NET classes to interact with these resources:

- [CosmosClient](#) - This class provides a client-side logical representation for the Azure Cosmos DB service. The client object is used to configure and execute requests against the service.
- [CreateDatabaseIfNotExistsAsync](#) - This method creates (if doesn't exist) or gets (if already exists) a database resource as an asynchronous operation.
- [CreateContainerIfNotExistsAsync](#) - This method creates (if it doesn't exist) or gets (if it already exists) a container as an asynchronous operation. You can check the status code from the response to determine whether the container was newly created (201) or an existing container was returned (200).
- [CreateItemAsync](#) - This method creates an item within the container.
- [UpsertItemAsync](#) - This method creates an item within the container if it doesn't already exist or replaces the item if it already exists.
- [GetItemQueryIterator](#) - This method creates a query for items under a container in an Azure Cosmos database using a SQL statement with parameterized values.
- [DeleteAsync](#) - Deletes the specified database from your Azure Cosmos account. `DeleteAsync` method only deletes the database. Disposing of the `CosmosClient` instance should happen separately (which it does in the `DeleteDatabaseAndCleanupAsync` method).

Code examples

The sample code described in this article creates a family database in Azure Cosmos DB. The family database contains family details such as name, address, location, the associated parents, children, and pets. Before populating the data to your Azure Cosmos account, define the properties of a family item. Create a new class named `Family.cs` at the root level of your sample application and add the following code to it:

```

using Newtonsoft.Json;

namespace todo
{
    public class Family
    {
        [JsonProperty(PropertyName = "id")]
        public string Id { get; set; }
        public string LastName { get; set; }
        public Parent[] Parents { get; set; }
        public Child[] Children { get; set; }
        public Address Address { get; set; }
        public bool IsRegistered { get; set; }
        // The ToString() method is used to format the output, it's used for demo purpose only. It's not
        required by Azure Cosmos DB
        public override string ToString()
        {
            return JsonConvert.SerializeObject(this);
        }
    }

    public class Parent
    {
        public string FamilyName { get; set; }
        public string FirstName { get; set; }
    }

    public class Child
    {
        public string FamilyName { get; set; }
        public string FirstName { get; set; }
        public string Gender { get; set; }
        public int Grade { get; set; }
        public Pet[] Pets { get; set; }
    }

    public class Pet
    {
        public string GivenName { get; set; }
    }

    public class Address
    {
        public string State { get; set; }
        public string County { get; set; }
        public string City { get; set; }
    }
}

```

Add the using directives & define the client object

From the project directory, open the `Program.cs` file in your editor and add the following using directives at the top of your application:

```

using System;
using System.Threading.Tasks;
using System.Configuration;
using System.Collections.Generic;
using System.Net;
using Microsoft.Azure.Cosmos;

```

To the `Program.cs` file, add code to read the environment variables that you have set in the previous step. Define the `CosmosClient`, `Database`, and the `Container` objects. Next add code to the main method that calls the

`GetStartedDemoAsync` method where you manage Azure Cosmos account resources.

```
namespace todo
{
    public class Program
    {

        /// The Azure Cosmos DB endpoint for running this GetStarted sample.
        private string EndpointUrl = Environment.GetEnvironmentVariable("EndpointUrl");

        /// The primary key for the Azure DocumentDB account.
        private string PrimaryKey = Environment.GetEnvironmentVariable("PrimaryKey");

        // The Cosmos client instance
        private CosmosClient cosmosClient;

        // The database we will create
        private Database database;

        // The container we will create.
        private Container container;

        // The name of the database and container we will create
        private string databaseId = "FamilyDatabase";
        private string containerId = "FamilyContainer";

        public static async Task Main(string[] args)
        {
            try
            {
                Console.WriteLine("Beginning operations...\n");
                Program p = new Program();
                await p.GetStartedDemoAsync();
            }
            catch (CosmosException de)
            {
                Exception baseException = de.GetBaseException();
                Console.WriteLine("{0} error occurred: {1}", de.StatusCode, de);
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: {0}", e);
            }
            finally
            {
                Console.WriteLine("End of demo, press any key to exit.");
                Console.ReadKey();
            }
        }
    }
}
```

Create a database

Define the `CreateDatabaseAsync` method within the `program.cs` class. This method creates the `FamilyDatabase` if it doesn't already exist.

```
private async Task CreateDatabaseAsync()
{
    // Create a new database
    this.database = await this.cosmosClient.CreateDatabaseIfNotExistsAsync(databaseId);
    Console.WriteLine("Created Database: {0}\n", this.database.Id);
}
```

Create a container

Define the `CreateContainerAsync` method within the `program.cs` class. This method creates the `FamilyContainer` if it doesn't already exist.

```
/// Create the container if it does not exist.
/// Specify "/LastName" as the partition key since we're storing family information, to ensure good
distribution of requests and storage.
private async Task CreateContainerAsync()
{
    // Create a new container
    this.container = await this.database.CreateContainerIfNotExistsAsync(containerId, "/LastName");
    Console.WriteLine("Created Container: {0}\n", this.container.Id);
}
```

Create an item

Create a family item by adding the `AddItemsToContainerAsync` method with the following code. You can use the `CreateItemAsync` or `UpsertItemAsync` methods to create an item:

```

private async Task AddItemsToContainerAsync()
{
    // Create a family object for the Andersen family
    Family andersenFamily = new Family
    {
        Id = "Andersen.1",
        LastName = "Andersen",
        Parents = new Parent[]
        {
            new Parent { FirstName = "Thomas" },
            new Parent { FirstName = "Mary Kay" }
        },
        Children = new Child[]
        {
            new Child
            {
                FirstName = "Henriette Thaulow",
                Gender = "female",
                Grade = 5,
                Pets = new Pet[]
                {
                    new Pet { GivenName = "Fluffy" }
                }
            }
        },
        Address = new Address { State = "WA", County = "King", City = "Seattle" },
        IsRegistered = false
    };

    try
    {
        // Create an item in the container representing the Andersen family. Note we provide the value of
        the partition key for this item, which is "Andersen".
        ItemResponse<Family> andersenFamilyResponse = await this.container.CreateItemAsync<Family>
        (andersenFamily, new PartitionKey(andersenFamily.LastName));
        // Note that after creating the item, we can access the body of the item with the Resource property
        of the ItemResponse. We can also access the RequestCharge property to see the amount of RUs consumed on this
        request.
        Console.WriteLine("Created item in database with id: {0} Operation consumed {1} RUs.\n",
        andersenFamilyResponse.Resource.Id, andersenFamilyResponse.RequestCharge);
    }
    catch (CosmosException ex) when (ex.StatusCode == HttpStatusCode.Conflict)
    {
        Console.WriteLine("Item in database with id: {0} already exists\n", andersenFamily.Id);
    }
}

```

Query the items

After inserting an item, you can run a query to get the details of "Andersen" family. The following code shows how to execute the query using the SQL query directly. The SQL query to get the "Anderson" family details is:

```
SELECT * FROM c WHERE c.LastName = 'Andersen'
```

Define the `QueryItemsAsync` method within the `program.cs`

class and add the following code to it:

```

private async Task QueryItemsAsync()
{
    var sqlQueryText = "SELECT * FROM c WHERE c.LastName = 'Andersen'";

    Console.WriteLine("Running query: {0}\n", sqlQueryText);

    QueryDefinition queryDefinition = new QueryDefinition(sqlQueryText);
    FeedIterator<Family> queryResultSetIterator = this.container.GetItemQueryIterator<Family>
(queryDefinition);

    List<Family> families = new List<Family>();

    while (queryResultSetIterator.HasMoreResults)
    {
        FeedResponse<Family> currentResultSet = await queryResultSetIterator.ReadNextAsync();
        foreach (Family family in currentResultSet)
        {
            families.Add(family);
            Console.WriteLine("\tRead {0}\n", family);
        }
    }
}

```

Delete the database

Finally you can delete the database adding the `DeleteDatabaseAndCleanupAsync` method with the following code:

```

private async Task DeleteDatabaseAndCleanupAsync()
{
    DatabaseResponse databaseResourceResponse = await this.database.DeleteAsync();
    // Also valid: await this.cosmosClient.Databases["FamilyDatabase"].DeleteAsync();

    Console.WriteLine("Deleted Database: {0}\n", this.databaseId);

    //Dispose of CosmosClient
    this.cosmosClient.Dispose();
}

```

Execute the CRUD operations

After you have defined all the required methods, execute them with in the `GetStartedDemoAsync` method. The `DeleteDatabaseAndCleanupAsync` method commented out in this code because you will not see any resources if that method is executed. You can uncomment it after validating that your Azure Cosmos DB resources were created in the Azure portal.

```

public async Task GetStartedDemoAsync()
{
    // Create a new instance of the Cosmos Client
    this.cosmosClient = new CosmosClient(EndpointUrl, PrimaryKey);
    await this.CreateDatabaseAsync();
    await this.CreateContainerAsync();
    await this.AddItemToContainerAsync();
    await this.QueryItemsAsync();
}

```

After you add all the required methods, save the `Program.cs` file.

Run the code

Next build and run the application to create the Azure Cosmos DB resources. Make sure to open a new

command prompt window, don't use the same instance that you have used to set the environment variables. Because the environment variables are not set in the current open window. You will need to open a new command prompt to see the updates.

```
dotnet build
```

```
dotnet run
```

The following output is generated when you run the application. You can also sign into the Azure portal and validate that the resources are created:

```
Created Database: FamilyDatabase

Created Container: FamilyContainer

Created item in database with id: Andersen.1 Operation consumed 11.62 RUs.

Running query: SELECT * FROM c WHERE c.LastName = 'Andersen'

      Read {"id":"Andersen.1","LastName":"Andersen","Parents":[{"FamilyName":null,"FirstName":"Thomas"},
{"FamilyName":null,"FirstName":"Mary Kay"}],"Children":[{"FamilyName":null,"FirstName":"Henriette
Thaulow","Gender":"female","Grade":5,"Pets":[{"GivenName":"Fluffy"}]}],"Address":
{"State":"WA","County":"King","City":"Seattle"},"IsRegistered":false}

End of demo, press any key to exit.
```

You can validate that the data is created by signing into the Azure portal and see the required items in your Azure Cosmos account.

Clean up resources

When no longer needed, you can use the Azure CLI or Azure PowerShell to remove the Azure Cosmos account and the corresponding resource group. The following command shows how to delete the resource group by using the Azure CLI:

```
az group delete -g "myResourceGroup"
```

Next steps

In this quickstart, you learned how to create an Azure Cosmos account, create a database and a container using a .NET Core app. You can now import additional data to your Azure Cosmos account with the instructions in the following article.

[Import data into Azure Cosmos DB](#)

Quickstart: Create an Azure SQL Database single database

5/28/2021 • 7 minutes to read • [Edit Online](#)

In this quickstart, you create a [single database](#) in Azure SQL Database using either the Azure portal, a PowerShell script, or an Azure CLI script. You then query the database using **Query editor** in the Azure portal.

Prerequisite

- An active Azure subscription. If you don't have one, [create a free account](#).

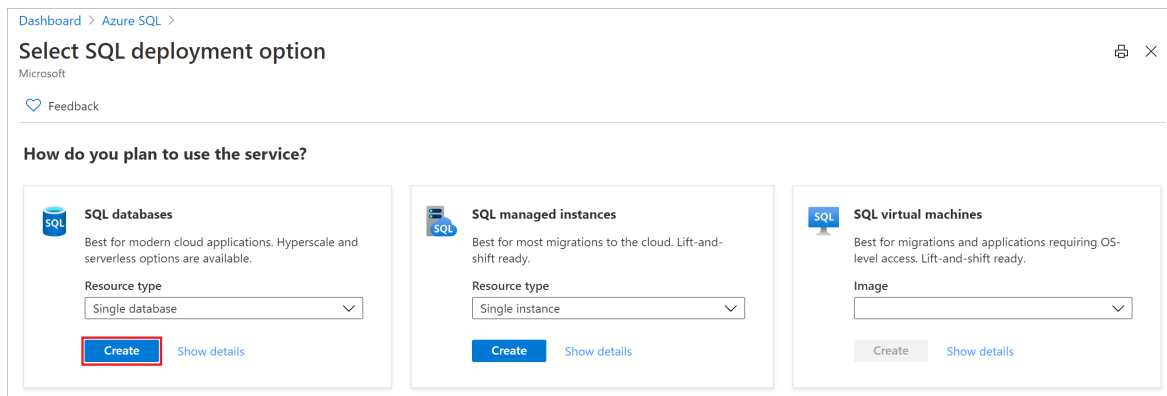
Create a single database

This quickstart creates a single database in the [serverless compute tier](#).

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

To create a single database in the Azure portal this quickstart starts at the Azure SQL page.

1. Browse to the [Select SQL Deployment option](#) page.
2. Under **SQL databases**, leave **Resource type** set to **Single database**, and select **Create**.



The screenshot shows the 'Select SQL deployment option' page in the Azure portal. The page is titled 'Select SQL deployment option' and has a breadcrumb 'Dashboard > Azure SQL >'. Below the title, there is a 'Feedback' link. The main content area is titled 'How do you plan to use the service?' and contains three cards:

- SQL databases**: Best for modern cloud applications. Hyperscale and serverless options are available. Resource type: Single database. Create button is highlighted with a red box.
- SQL managed instances**: Best for most migrations to the cloud. Lift-and-shift ready. Resource type: Single instance.
- SQL virtual machines**: Best for migrations and applications requiring OS-level access. Lift-and-shift ready. Image: (empty dropdown).

3. On the **Basics** tab of the **Create SQL Database** form, under **Project details**, select the desired **Azure Subscription**.
4. For **Resource group**, select **Create new**, enter *myResourceGroup*, and select **OK**.
5. For **Database name** enter *mySampleDatabase*.
6. For **Server**, select **Create new**, and fill out the **New server** form with the following values:
 - **Server name**: Enter *mysqlserver*, and add some characters for uniqueness. We can't provide an exact server name to use because server names must be globally unique for all servers in Azure, not just unique within a subscription. So enter something like *mysqlserver12345*, and the portal lets you know if it is available or not.
 - **Server admin login**: Enter *azureuser*.
 - **Password**: Enter a password that meets requirements, and enter it again in the **Confirm password**

field.

- **Location:** Select a location from the dropdown list.

Select OK.

7. Leave **Want to use SQL elastic pool** set to **No**.

8. Under **Compute + storage**, select **Configure database**.

9. This quickstart uses a serverless database, so select **Serverless**, and then select **Apply**.

The screenshot shows the 'Configure' page for an Azure SQL database. The page is titled 'Configure' and has a breadcrumb trail: 'Home > Azure SQL > Select SQL deployment option > Create SQL Database > Configure'. There are three tabs: 'General Purpose', 'Hyperscale', and 'Business Critical'. The 'General Purpose' tab is active, showing 'Scalable compute and storage options' with '500 - 20,000 IOPS' and '2-10 ms latency'. The 'Serverless' compute tier is selected, indicated by a blue checkmark and a red box. It is described as 'Compute resources are auto-scaled' and 'Billed per second based on vCores used'. The 'Compute Hardware' section shows 'Gen5' hardware with 'up to 40 vCores, up to 120 GB memory' and a 'Change configuration' link. The 'Max vCores' slider is set to 1, and the 'Min vCores' slider is set to 0.5. The 'Auto-pause delay' section has 'enable auto-pause' checked, with 'Days' set to 0, 'Hours' set to 1, and 'Minutes' set to 0. At the bottom, the 'Apply' button is highlighted with a red box. On the right side, there is a 'Cost summary' section showing 'Gen5 - General Purpose (GP, S, Gen5, T)' with 'Cost per GB' and 'Max storage selected (in GB)'. Below that, there are 'ESTIMATED STORAGE COST / MONTH' and 'COMPUTE COST / VCORE / SECOND' sections, and a 'NOTES' section stating 'Serverless databases are billed in vCores based on a combination of CPU and memory utilization. Learn more about serverless billing.'

10. Select **Next: Networking** at the bottom of the page.

Create SQL Database

Microsoft


provision with smart defaults, or visit each tab to customize. [Learn more](#) 

Project details

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

Subscription * 

select your subscription 

Resource group * 

(New) myResourceGroup 

[Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources


Database name *

mySampleDatabase 


Server * 

(new) mysqlserver0819 (East US) 

[Create new](#)

Want to use SQL elastic pool? * 

Yes No

Compute + storage * 

General Purpose

Serverless, Gen5, 1 vCore, 32 GB storage

[Configure database](#)

[Review + create](#)

[Next : Networking >](#)

11. On the **Networking** tab, for **Connectivity method**, select **Public endpoint**.
12. For **Firewall rules**, set **Add current client IP address** to **Yes**. Leave **Allow Azure services and resources to access this server** set to **No**.
13. Select **Next: Additional settings** at the bottom of the page.

Create SQL Database

Microsoft

[Basics](#) **[Networking](#)** [Additional settings](#) [Tags](#) [Review + create](#)

Configure network access and connectivity for your server. The configuration selected below will apply to the selected server 'mysqlserver-12' and all databases it manages. [Learn more](#)

Network connectivity

Choose an option for configuring connectivity to your server via public endpoint or private endpoint. Choosing no access creates with defaults and you can configure connection method after server creation. [Learn more](#)

Connectivity method *

No access

Public endpoint

Private endpoint (preview)

Firewall rules

Setting 'Allow Azure services and resources to access this server' to Yes allows communications from all resources inside the Azure boundary, that may or may not be part of your subscription. [Learn more](#)

Setting 'Add current client IP address' to Yes will add an entry for your client IP address to the server firewall.

Allow Azure services and resources to access this server * No Yes

Add current client IP address * No Yes

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

14. On the **Additional settings** tab, in the **Data source** section, for **Use existing data**, select **Sample**. This creates an AdventureWorksLT sample database so there's some tables and data to query and experiment with, as opposed to an empty blank database.
15. Optionally, enable [Azure Defender for SQL](#).
16. Optionally, set the [maintenance window](#) so planned maintenance is performed at the best time for your database.
17. Select **Review + create** at the bottom of the page:

[Home](#) > [servercontoso](#) >

Create SQL Database

Microsoft

[Basics](#) [Networking](#) [Additional settings](#) [Tags](#) [Review + create](#)

Customize additional configuration parameters including collation & sample data.

Data source

Start with a blank database, restore from a backup or select sample data to populate your new database.

Use existing data *

 None Backup Sample

AdventureWorksLT will be created as the sample database.

Database collation

Database collation defines the rules that sort and compare data, and cannot be changed after database creation. The default database collation is SQL_Latin1_General_CP1_CI_AS. [Learn more](#)

Collation ⓘ

SQL_Latin1_General_CP1_CI_AS

Azure Defender for SQL

Protect your data using Azure Defender for SQL, a unified security package including vulnerability assessment and advanced threat protection for your server. [Learn more](#)

Advanced Data Security costs 15 USD/server/month.

Enable Azure Defender for SQL * ⓘ

 Enable Not now

Maintenance window

Select a preferred maintenance window from the drop down. Please note, during a maintenance event, Azure SQL Database are fully available and accessible but some of the maintenance updates require a failover as Azure takes SQL DB instances offline for a short time to apply the maintenance updates. If the database is part of elastic pool, the maintenance configuration of elastic pool will be applied. [Learn more](#)

Maintenance window

System default

[Review + create](#)

< Previous

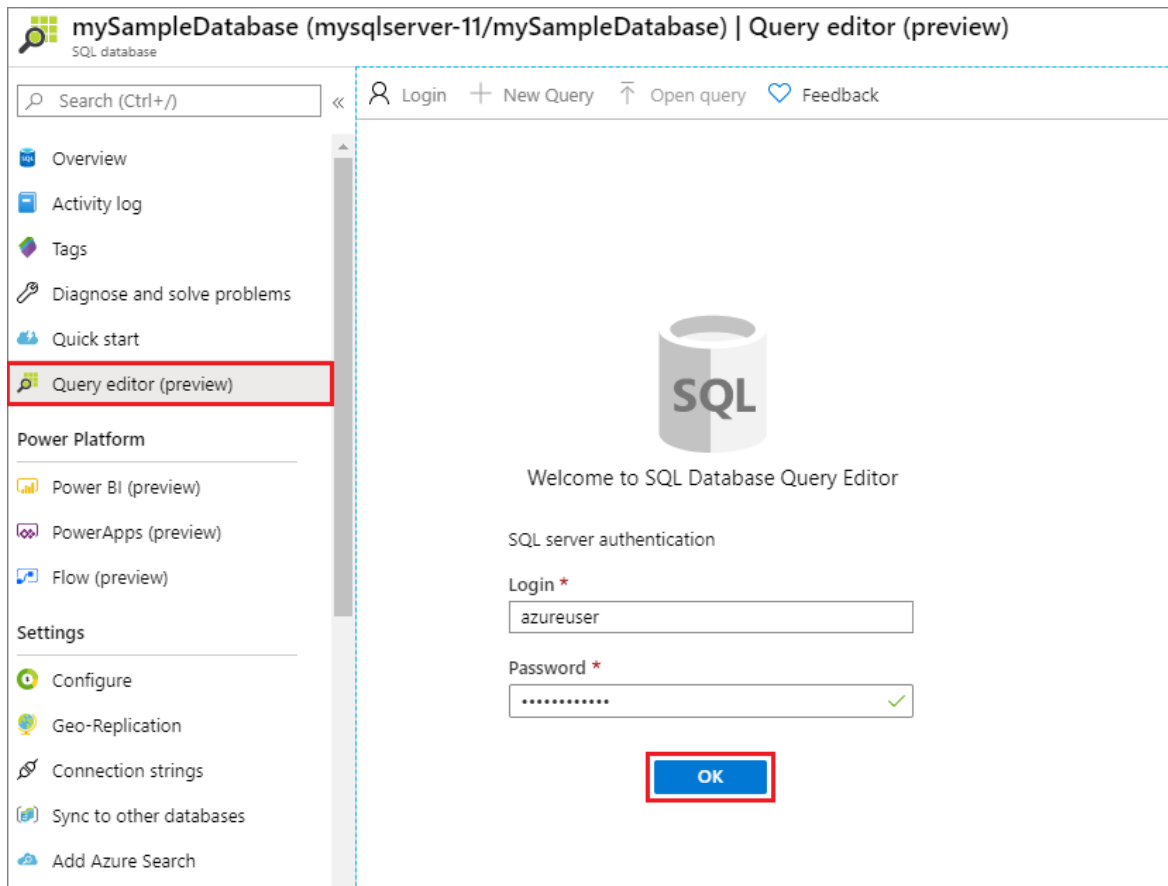
Next : Tags >

18. On the **Review + create** page, after reviewing, select **Create**.

Query the database

Once your database is created, you can use the **Query editor (preview)** in the Azure portal to connect to the database and query data.

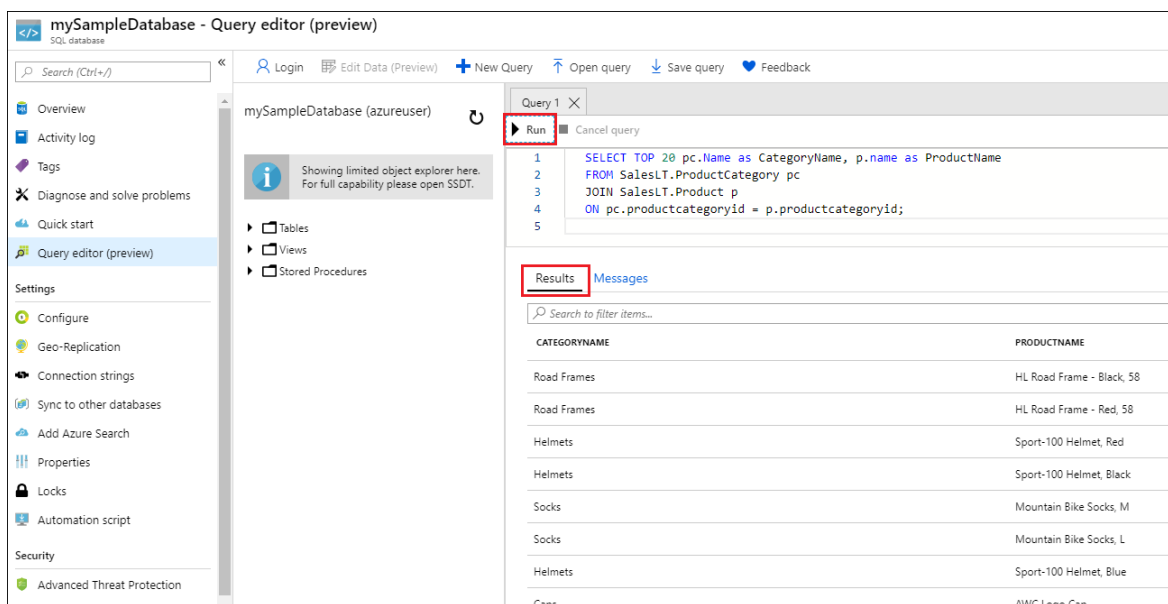
1. In the portal, search for and select **SQL databases**, and then select your database from the list.
2. On the page for your database, select **Query editor (preview)** in the left menu.
3. Enter your server admin login information, and select **OK**.



4. Enter the following query in the Query editor pane.

```
SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
FROM SalesLT.ProductCategory pc
JOIN SalesLT.Product p
ON pc.productcategoryid = p.productcategoryid;
```

5. Select Run, and then review the query results in the Results pane.



6. Close the Query editor page, and select OK when prompted to discard your unsaved edits.

Clean up resources

Keep the resource group, server, and single database to go on to the next steps, and learn how to connect and

query your database with different methods.

When you're finished using these resources, you can delete the resource group you created, which will also delete the server and single database within it.

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

To delete **myResourceGroup** and all its resources using the Azure portal:

1. In the portal, search for and select **Resource groups**, and then select **myResourceGroup** from the list.
2. On the resource group page, select **Delete resource group**.
3. Under **Type the resource group name**, enter *myResourceGroup*, and then select **Delete**.

Next steps

[Connect and query](#) your database using different tools and languages:

[Connect and query using SQL Server Management Studio](#)

[Connect and query using Azure Data Studio](#)

Want to optimize and save on your cloud spending?

[Start analyzing costs with Cost Management](#)

Get started with Azure Queue Storage using .NET

5/25/2021 • 20 minutes to read • [Edit Online](#)

Overview

Azure Queue Storage provides cloud messaging between application components. In designing applications for scale, application components are often decoupled so they can scale independently. Queue Storage delivers asynchronous messaging between application components, whether they are running in the cloud, on the desktop, on an on-premises server, or on a mobile device. Queue Storage also supports managing asynchronous tasks and building process work flows.

About this tutorial

This tutorial shows how to write .NET code for some common scenarios using Azure Queue Storage. Scenarios covered include creating and deleting queues and adding, reading, and deleting queue messages.

Estimated time to complete: 45 minutes

Prerequisites

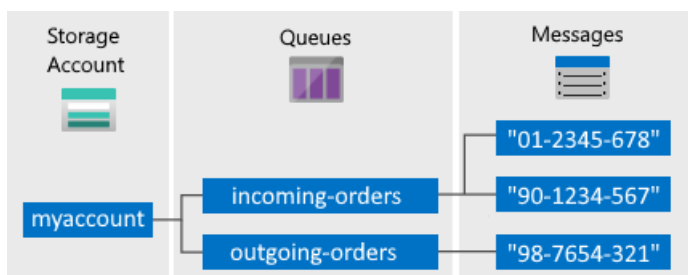
- [Microsoft Visual Studio](#)
- An [Azure Storage account](#)

What is Queue storage?

Azure Queue storage is a service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account. Queue storage is often used to create a backlog of work to process asynchronously.

Queue service concepts

The Azure Queue service contains the following components:



- **Storage Account:** All access to Azure Storage is done through a storage account. For more information about storage accounts, see [Storage account overview](#).
- **Queue:** A queue contains a set of messages. All messages must be in a queue. Note that the queue name must be all lowercase. For information on naming queues, see [Naming Queues and Metadata](#).
- **Message:** A message, in any format, of up to 64 KB. The maximum time that a message can remain in the queue is 7 days. For version 2017-07-29 or later, the maximum time-to-live can be any positive number, or -1 indicating that the message doesn't expire. If this parameter is omitted, the default time-to-live is seven days.
- **URL format:** Queues are addressable using the following URL format: `http://<storage account>`

.queue.core.windows.net/<queue>

The following URL addresses a queue in the diagram:

```
http://myaccount.queue.core.windows.net/incoming-orders
```

Create an Azure storage account

The easiest way to create your first Azure storage account is by using the [Azure portal](#). To learn more, see [Create a storage account](#).

You can also create an Azure storage account by using [Azure PowerShell](#), [Azure CLI](#), or the [Azure Storage Resource Provider for .NET](#).

If you prefer not to create a storage account in Azure at this time, you can also use the Azurite storage emulator to run and test your code in a local environment. For more information, see [Use the Azurite emulator for local Azure Storage development](#).

Set up your development environment

Next, set up your development environment in Visual Studio so you're ready to try the code examples in this guide.

Create a Windows console application project

In Visual Studio, create a new Windows console application. The following steps show you how to create a console application in Visual Studio 2019. The steps are similar in other versions of Visual Studio.

1. Select **File > New > Project**
2. Select **Platform > Windows**
3. Select **Console App (.NET Framework)**
4. Select **Next**
5. In the **Project name** field, enter a name for your application
6. Select **Create**

All code examples in this tutorial can be added to the `Main()` method of your console application's `Program.cs` file.

You can use the Azure Storage client libraries in any type of .NET application, including an Azure cloud service or web app, and desktop and mobile applications. In this guide, we use a console application for simplicity.

Use NuGet to install the required packages

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

You need to reference the following four packages in your project to complete this tutorial:

- [Azure.Core library for .NET](#): This package provides shared primitives, abstractions, and helpers for modern .NET Azure SDK client libraries.
- [Azure.Storage.Common client library for .NET](#): This package provides infrastructure shared by the other Azure Storage client libraries.
- [Azure.Storage.Queues client library for .NET](#): This package enables working with Azure Queue Storage for storing messages that may be accessed by a client.
- [System.Configuration.ConfigurationManager library for .NET](#): This package provides access to configuration files for client applications.

You can use NuGet to obtain these packages. Follow these steps:

1. Right-click your project in **Solution Explorer**, and choose **Manage NuGet Packages**.
2. Select **Browse**
3. Search online for `Azure.Storage.Queues`, and select **Install** to install the Azure Storage client library and its dependencies. This will also install the `Azure.Storage.Common` and `Azure.Core` libraries, which are dependencies of the queue library.
4. Search online for `System.Configuration.ConfigurationManager`, and select **Install** to install the Configuration Manager.

Determine your target environment

You have two environment options for running the examples in this guide:

- You can run your code against an Azure Storage account in the cloud.
- You can run your code against the Azurite storage emulator. Azurite is a local environment that emulates an Azure Storage account in the cloud. Azurite is a free option for testing and debugging your code while your application is under development. The emulator uses a well-known account and key. For more information, see [Use the Azurite emulator for local Azure Storage development and testing](#).

NOTE

You can target the storage emulator to avoid incurring any costs associated with Azure Storage. However, if you do choose to target an Azure Storage account in the cloud, costs for performing this tutorial will be negligible.

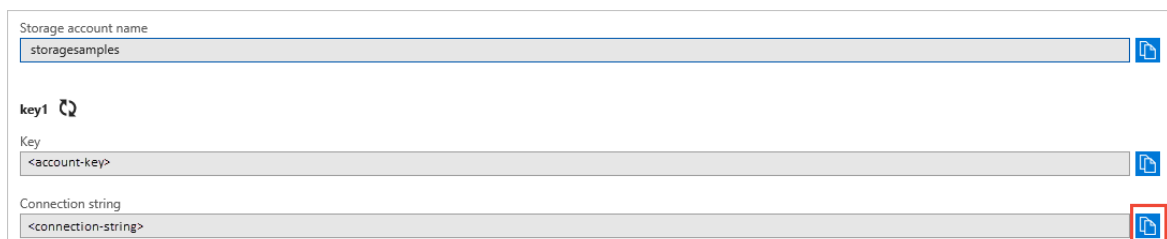
Get your storage connection string

The Azure Storage client libraries for .NET support using a storage connection string to configure endpoints and credentials for accessing storage services. For more information, see [Manage storage account access keys](#).

Copy your credentials from the Azure portal

The sample code needs to authorize access to your storage account. To authorize, you provide the application with your storage account credentials in the form of a connection string. To view your storage account credentials:

1. Navigate to the [Azure portal](#).
2. Locate your storage account.
3. In the **Settings** section of the storage account overview, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
4. Find the **Connection string** value under **key1**, and click the **Copy** button to copy the connection string. You will add the connection string value to an environment variable in the next step.



The screenshot shows the 'Access keys' section of an Azure Storage account. It displays three rows of information:

- Storage account name:** storagesamples
- key1:** Key: <account-key>
- Connection string:** <connection-string>

The 'Connection string' field is highlighted with a red box, and a red square icon (the copy button) is visible at the end of the field.

For more information about connection strings, see [Configure a connection string to Azure Storage](#).

NOTE

Your storage account key is similar to the root password for your storage account. Always be careful to protect your storage account key. Avoid distributing it to other users, hard-coding it, or saving it in a plain-text file that is accessible to others. Regenerate your key by using the Azure portal if you believe it may have been compromised.

The best way to maintain your storage connection string is in a configuration file. To configure your connection string, open the `app.config` file from Solution Explorer in Visual Studio. Add the contents of the `<appSettings>` element shown here. Replace `connection-string` with the value you copied from your storage account in the portal:

```
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <appSettings>
    <add key="StorageConnectionString" value="connection-string" />
  </appSettings>
</configuration>
```

For example, your configuration setting appears similar to:

```
<add key="StorageConnectionString"
value="DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=GMuzNHj1B3S9itqZJHHCnRkrokLkcSyw7
yK9BRbGp0ENePunLPwBgpxV1Z/pVo9zpem/2xSHXkMqTHHLcx8XRA==EndpointSuffix=core.windows.net" />
```

To target the Azurite storage emulator, you can use a shortcut that maps to the well-known account name and key. In that case, your connection string setting is:

```
<add key="StorageConnectionString" value="UseDevelopmentStorage=true" />
```

Add using directives

Add the following `using` directives to the top of the `Program.cs` file:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
using System; // Namespace for Console output
using System.Configuration; // Namespace for ConfigurationManager
using System.Threading.Tasks; // Namespace for Task
using Azure.Storage.Queues; // Namespace for Queue storage types
using Azure.Storage.Queues.Models; // Namespace for PeekedMessage
```

Create the Queue Storage client

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

The `QueueClient` class enables you to retrieve queues stored in Queue Storage. Here's one way to create the service client:


```
//-----
// Create the queue service client
//-----
public void CreateQueueClient(string queueName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a QueueClient which will be used to create and manipulate the queue
    QueueClient queueClient = new QueueClient(connectionString, queueName);
}
}
```

Now you are ready to write code that reads data from and writes data to Queue Storage.

Create a queue

This example shows how to create a queue:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
//-----
// Create a message queue
//-----
public bool CreateQueue(string queueName)
{
    try
    {
        // Get the connection string from app settings
        string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

        // Instantiate a QueueClient which will be used to create and manipulate the queue
        QueueClient queueClient = new QueueClient(connectionString, queueName);

        // Create the queue
        queueClient.CreateIfNotExists();

        if (queueClient.Exists())
        {
            Console.WriteLine($"Queue created: '{queueClient.Name}'");
            return true;
        }
        else
        {
            Console.WriteLine($"Make sure the Azurite storage emulator running and try again.");
            return false;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception: {ex.Message}\n\n");
        Console.WriteLine($"Make sure the Azurite storage emulator running and try again.");
        return false;
    }
}
}
```

Insert a message into a queue

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To insert a message into an existing queue, call the `SendMessage` method. A message can be either a string (in UTF-8 format) or a byte array. The following code creates a queue (if it doesn't exist) and inserts a message:

```
//-----  
// Insert a message into a queue  
//-----  
public void InsertMessage(string queueName, string message)  
{  
    // Get the connection string from app settings  
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];  
  
    // Instantiate a QueueClient which will be used to create and manipulate the queue  
    QueueClient queueClient = new QueueClient(connectionString, queueName);  
  
    // Create the queue if it doesn't already exist  
    queueClient.CreateIfNotExists();  
  
    if (queueClient.Exists())  
    {  
        // Send a message to the queue  
        queueClient.SendMessage(message);  
    }  
  
    Console.WriteLine($"Inserted: {message}");  
}
```

Peek at the next message

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

You can peek at the messages in the queue without removing them from the queue by calling the `PeekMessages` method. If you don't pass a value for the `maxMessages` parameter, the default is to peek at one message.

```
//-----  
// Peek at a message in the queue  
//-----  
public void PeekMessage(string queueName)  
{  
    // Get the connection string from app settings  
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];  
  
    // Instantiate a QueueClient which will be used to manipulate the queue  
    QueueClient queueClient = new QueueClient(connectionString, queueName);  
  
    if (queueClient.Exists())  
    {  
        // Peek at the next message  
        PeekedMessage[] peekedMessage = queueClient.PeekMessages();  
  
        // Display the message  
        Console.WriteLine($"Peeked message: '{peekedMessage[0].MessageText}'");  
    }  
}
```

Change the contents of a queued message

You can change the contents of a message in-place in the queue. If the message represents a work task, you could use this feature to update the status of the work task. The following code updates the queue message with new contents, and sets the visibility timeout to extend another 60 seconds. This saves the state of work

associated with the message, and gives the client another minute to continue working on the message. You could use this technique to track multistep workflows on queue messages, without having to start over from the beginning if a processing step fails due to hardware or software failure. Typically, you would keep a retry count as well, and if the message is retried more than n times, you would delete it. This protects against a message that triggers an application error each time it is processed.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
//-----  
// Update an existing message in the queue  
//-----  
public void UpdateMessage(string queueName)  
{  
    // Get the connection string from app settings  
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];  
  
    // Instantiate a QueueClient which will be used to manipulate the queue  
    QueueClient queueClient = new QueueClient(connectionString, queueName);  
  
    if (queueClient.Exists())  
    {  
        // Get the message from the queue  
        QueueMessage[] message = queueClient.ReceiveMessages();  
  
        // Update the message contents  
        queueClient.UpdateMessage(message[0].MessageId,  
            message[0].PopReceipt,  
            "Updated contents",  
            TimeSpan.FromSeconds(60.0) // Make it invisible for another 60 seconds  
        );  
    }  
}
```

Dequeue the next message

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

Dequeue a message from a queue in two steps. When you call `ReceiveMessages`, you get the next message in a queue. A message returned from `ReceiveMessages` becomes invisible to any other code reading messages from this queue. By default, this message stays invisible for 30 seconds. To finish removing the message from the queue, you must also call `DeleteMessage`. This two-step process of removing a message assures that if your code fails to process a message due to hardware or software failure, another instance of your code can get the same message and try again. Your code calls `DeleteMessage` right after the message has been processed.

```
//-----
// Process and remove a message from the queue
//-----
public void DequeueMessage(string queueName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a QueueClient which will be used to manipulate the queue
    QueueClient queueClient = new QueueClient(connectionString, queueName);

    if (queueClient.Exists())
    {
        // Get the next message
        QueueMessage[] retrievedMessage = queueClient.ReceiveMessages();

        // Process (i.e. print) the message in less than 30 seconds
        Console.WriteLine($"Dequeued message: '{retrievedMessage[0].MessageText}'");

        // Delete the message
        queueClient.DeleteMessage(retrievedMessage[0].MessageId, retrievedMessage[0].PopReceipt);
    }
}
}
```

Use the Async-Await pattern with common Queue Storage APIs

This example shows how to use the Async-Await pattern with common Queue Storage APIs. The sample calls the asynchronous version of each of the given methods, as indicated by the `Async` suffix of each method. When an async method is used, the Async-Await pattern suspends local execution until the call completes. This behavior allows the current thread to do other work, which helps avoid performance bottlenecks and improves the overall responsiveness of your application. For more details on using the Async-Await pattern in .NET, see [Async and Await \(C# and Visual Basic\)](#)

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```

//-----
// Perform queue operations asynchronously
//-----
public async Task QueueAsync(string queueName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a QueueClient which will be used to manipulate the queue
    QueueClient queueClient = new QueueClient(connectionString, queueName);

    // Create the queue if it doesn't already exist
    await queueClient.CreateIfNotExistsAsync();

    if (await queueClient.ExistsAsync())
    {
        Console.WriteLine($"Queue '{queueClient.Name}' created");
    }
    else
    {
        Console.WriteLine($"Queue '{queueClient.Name}' exists");
    }

    // Async enqueue the message
    await queueClient.SendMessageAsync("Hello, World");
    Console.WriteLine($"Message added");

    // Async receive the message
    QueueMessage[] retrievedMessage = await queueClient.ReceiveMessagesAsync();
    Console.WriteLine($"Retrieved message with content '{retrievedMessage[0].MessageText}'");

    // Async delete the message
    await queueClient.DeleteMessageAsync(retrievedMessage[0].MessageId, retrievedMessage[0].PopReceipt);
    Console.WriteLine($"Deleted message: '{retrievedMessage[0].MessageText}'");

    // Async delete the queue
    await queueClient.DeleteAsync();
    Console.WriteLine($"Deleted queue: '{queueClient.Name}'");
}

```

Use additional options for dequeuing messages

There are two ways you can customize message retrieval from a queue. First, you can get a batch of messages (up to 32). Second, you can set a longer or shorter invisibility timeout, allowing your code more or less time to fully process each message.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

The following code example uses the `ReceiveMessages` method to get 20 messages in one call. Then it processes each message using a `foreach` loop. It also sets the invisibility timeout to five minutes for each message. Note that the five minutes starts for all messages at the same time, so after five minutes have passed since the call to `ReceiveMessages`, any messages which have not been deleted will become visible again.

```
//-----
// Process and remove multiple messages from the queue
//-----
public void DequeueMessages(string queueName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a QueueClient which will be used to manipulate the queue
    QueueClient queueClient = new QueueClient(connectionString, queueName);

    if (queueClient.Exists())
    {
        // Receive and process 20 messages
        QueueMessage[] receivedMessages = queueClient.ReceiveMessages(20, TimeSpan.FromMinutes(5));

        foreach (QueueMessage message in receivedMessages)
        {
            // Process (i.e. print) the messages in less than 5 minutes
            Console.WriteLine($"De-queued message: '{message.MessageText}'");

            // Delete the message
            queueClient.DeleteMessage(message.MessageId, message.PopReceipt);
        }
    }
}
}
```

Get the queue length

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

You can get an estimate of the number of messages in a queue. The `GetProperties` method returns queue properties including the message count. The `ApproximateMessagesCount` property contains the approximate number of messages in the queue. This number is not lower than the actual number of messages in the queue, but could be higher.

```
//-----
// Get the approximate number of messages in the queue
//-----
public void GetQueueLength(string queueName)
{
    // Get the connection string from app settings
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

    // Instantiate a QueueClient which will be used to manipulate the queue
    QueueClient queueClient = new QueueClient(connectionString, queueName);

    if (queueClient.Exists())
    {
        QueueProperties properties = queueClient.GetProperties();

        // Retrieve the cached approximate message count.
        int cachedMessagesCount = properties.ApproximateMessagesCount;

        // Display number of messages.
        Console.WriteLine($"Number of messages in queue: {cachedMessagesCount}");
    }
}
}
```

Delete a queue

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To delete a queue and all the messages contained in it, call the `Delete` method on the queue object.

```
//-----  
// Delete the queue  
//-----  
public void DeleteQueue(string queueName)  
{  
    // Get the connection string from app settings  
    string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];  
  
    // Instantiate a QueueClient which will be used to manipulate the queue  
    QueueClient queueClient = new QueueClient(connectionString, queueName);  
  
    if (queueClient.Exists())  
    {  
        // Delete the queue  
        queueClient.Delete();  
    }  
  
    Console.WriteLine($"Queue deleted: '{queueClient.Name}'");  
}
```

Next steps

Now that you've learned the basics of Queue Storage, follow these links to learn about more complex storage tasks.

- View the Queue Storage reference documentation for complete details about available APIs:
 - [Azure Storage client library for .NET reference](#)
 - [Azure Storage REST API reference](#)
- View more feature guides to learn about additional options for storing data in Azure.
 - [Get started with Azure Table Storage using .NET](#) to store structured data.
 - [Get started with Azure Blob Storage using .NET](#) to store unstructured data.
 - [Connect to SQL Database by using .NET \(C#\)](#) to store relational data.
- Learn how to simplify the code you write to work with Azure Storage by using the [Azure WebJobs SDK](#).

Scale up an app in Azure App Service

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to scale your app in Azure App Service. There are two workflows for scaling, scale up and scale out, and this article explains the scale up workflow.

- **Scale up:** Get more CPU, memory, disk space, and extra features like dedicated virtual machines (VMs), custom domains and certificates, staging slots, autoscaling, and more. You scale up by changing the pricing tier of the App Service plan that your app belongs to.
- **Scale out:** Increase the number of VM instances that run your app. You can scale out to as many as 30 instances, depending on your pricing tier. [App Service Environments](#) in **Isolated** tier further increases your scale-out count to 100 instances. For more information about scaling out, see [Scale instance count manually or automatically](#). There, you find out how to use autoscaling, which is to scale instance count automatically based on predefined rules and schedules.

The scale settings take only seconds to apply and affect all apps in your [App Service plan](#). They don't require you to change your code or redeploy your application.

For information about the pricing and features of individual App Service plans, see [App Service Pricing Details](#).

NOTE

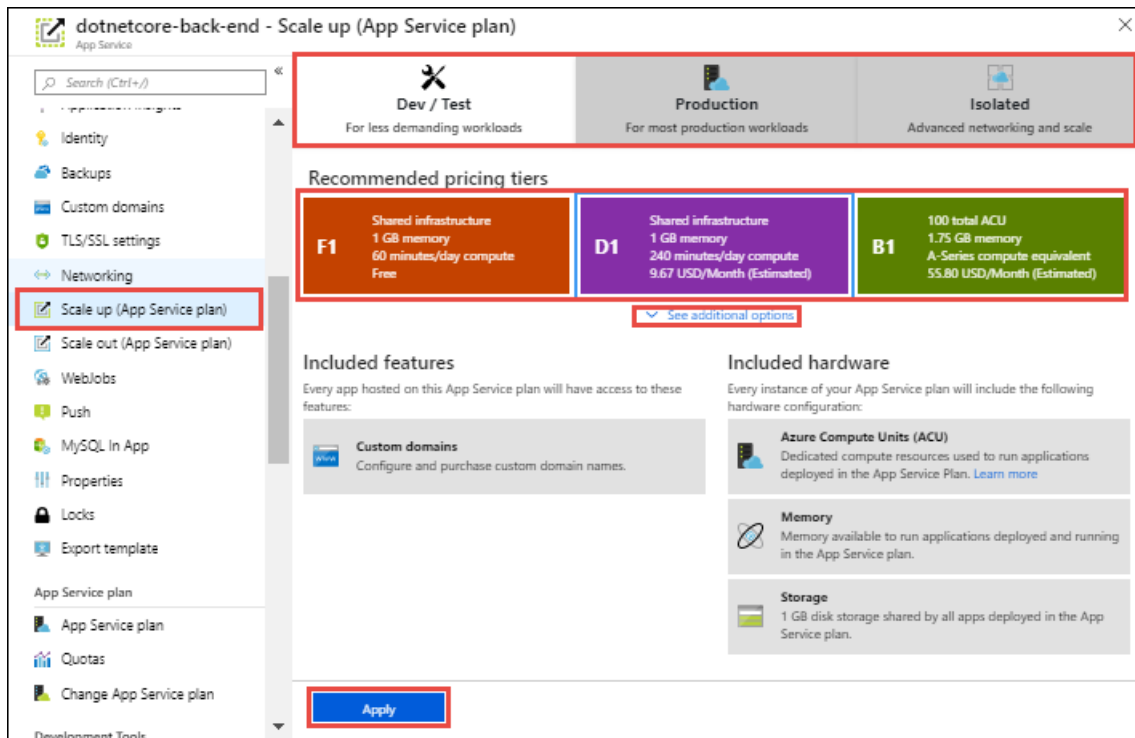
Before you switch an App Service plan from the **Free** tier, you must first remove the [spending limits](#) in place for your Azure subscription. To view or change options for your Microsoft Azure App Service subscription, see [Microsoft Azure Subscriptions](#).

Scale up your pricing tier

NOTE

To scale up to **PremiumV3** tier, see [Configure PremiumV3 tier for App Service](#).

1. In your browser, open the [Azure portal](#).
2. In your App Service app page, from the left menu, select **Scale Up (App Service plan)**.
3. Choose your tier, and then select **Apply**. Select the different categories (for example, **Production**) and also **See additional options** to show more tiers.

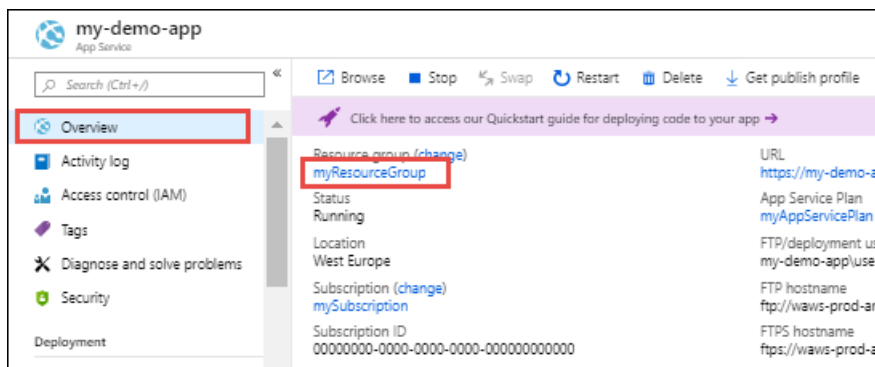


When the operation is complete, you see a notification pop-up with a green success check mark.

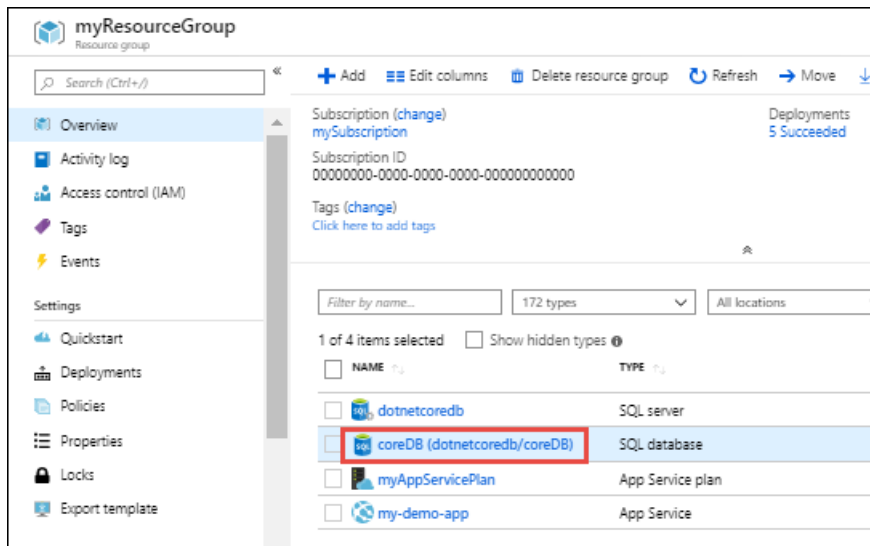
Scale related resources

If your app depends on other services, such as Azure SQL Database or Azure Storage, you can scale up these resources separately. These resources aren't managed by the App Service plan.

1. In the **Overview** page for your app, select the **Resource group** link.



2. In the **Summary** part of the **Resource group** page, select a resource that you want to scale. The following screenshot shows a SQL Database resource.



To scale up the related resource, see the documentation for the specific resource type. For example, to scale up a single SQL Database, see [Scale single database resources in Azure SQL Database](#). To scale up a Azure Database for MySQL resource, see [Scale MySQL resources](#).

Compare pricing tiers

For detailed information, such as VM sizes for each pricing tier, see [App Service Pricing Details](#).

For a table of service limits, quotas, and constraints, and supported features in each tier, see [App Service limits](#).

More resources

[Scale instance count manually or automatically](#)

[Configure PremiumV3 tier for App Service](#)

What are virtual machine scale sets?

6/17/2021 • 4 minutes to read • [Edit Online](#)

Azure virtual machine scale sets let you create and manage a group of load balanced VMs. The number of VM instances can automatically increase or decrease in response to demand or a defined schedule. Scale sets provide high availability to your applications, and allow you to centrally manage, configure, and update a large number of VMs. With virtual machine scale sets, you can build large-scale services for areas such as compute, big data, and container workloads.

Why use virtual machine scale sets?

To provide redundancy and improved performance, applications are typically distributed across multiple instances. Customers may access your application through a load balancer that distributes requests to one of the application instances. If you need to perform maintenance or update an application instance, your customers must be distributed to another available application instance. To keep up with extra customer demand, you may need to increase the number of application instances that run your application.

Azure virtual machine scale sets provide the management capabilities for applications that run across many VMs, [automatic scaling of resources](#), and load balancing of traffic. Scale sets provide the following key benefits:

- **Easy to create and manage multiple VMs**
 - When you have many VMs that run your application, it's important to maintain a consistent configuration across your environment. For reliable performance of your application, the VM size, disk configuration, and application installs should match across all VMs.
 - With scale sets, all VM instances are created from the same base OS image and configuration. This approach lets you easily manage hundreds of VMs without extra configuration tasks or network management.
 - Scale sets support the use of the [Azure load balancer](#) for basic layer-4 traffic distribution, and [Azure Application Gateway](#) for more advanced layer-7 traffic distribution and TLS termination.
- **Provides high availability and application resiliency**
 - Scale sets are used to run multiple instances of your application. If one of these VM instances has a problem, customers continue to access your application through one of the other VM instances with minimal interruption.
 - For more availability, you can use [Availability Zones](#) to automatically distribute VM instances in a scale set within a single datacenter or across multiple datacenters.
- **Allows your application to automatically scale as resource demand changes**
 - Customer demand for your application may change throughout the day or week. To match customer demand, scale sets can automatically increase the number of VM instances as application demand increases, then reduce the number of VM instances as demand decreases.
 - Autoscale also minimizes the number of unnecessary VM instances that run your application when demand is low, while customers continue to receive an acceptable level of performance as demand grows and additional VM instances are automatically added. This ability helps reduce costs and efficiently create Azure resources as required.
- **Works at large-scale**
 - Scale sets support up to 1,000 VM instances for standard marketplace images and custom images through the Shared Image Gallery. If you create a scale set using a managed image, the limit is 600

VM instances.

- For the best performance with production workloads, use [Azure Managed Disks](#).

Differences between virtual machines and scale sets

Scale sets are built from virtual machines. With scale sets, the management and automation layers are provided to run and scale your applications. You could instead manually create and manage individual VMs, or integrate existing tools to build a similar level of automation. The following table outlines the benefits of scale sets compared to manually managing multiple VM instances.

SCENARIO	MANUAL GROUP OF VMS	VIRTUAL MACHINE SCALE SET
Add extra VM instances	Manual process to create, configure, and ensure compliance	Automatically create from central configuration
Traffic balancing and distribution	Manual process to create and configure Azure load balancer or Application Gateway	Can automatically create and integrate with Azure load balancer or Application Gateway
High availability and redundancy	Manually create Availability Set or distribute and track VMs across Availability Zones	Automatic distribution of VM instances across Availability Zones or Availability Sets
Scaling of VMs	Manual monitoring and Azure Automation	Autoscale based on host metrics, in-guest metrics, Application Insights, or schedule

There is no extra cost to scale sets. You only pay for the underlying compute resources such as the VM instances, load balancer, or Managed Disk storage. The management and automation features, such as autoscale and redundancy, incur no additional charges over the use of VMs.

How to monitor your scale sets

Use [Azure Monitor for VMs](#), which has a simple onboarding process and will automate the collection of important CPU, memory, disk, and network performance counters from the VMs in your scale set. It also includes extra monitoring capabilities and pre-defined visualizations that help you focus on the availability and performance of your scale sets.

Enable monitoring for your [virtual machine scale set application](#) with Application Insights to collect detailed information about your application including page views, application requests, and exceptions. Further verify the availability of your application by configuring an [availability test](#) to simulate user traffic.

Data residency

In Azure, the feature to enable storing customer data in a single region is currently only available in the Southeast Asia Region (Singapore) of the Asia Pacific Geo and Brazil South (Sao Paulo State) Region of Brazil Geo. Customer data is stored in Geo for all other regions. See [Trust Center](#) for more information.

Next steps

To get started, create your first virtual machine scale set in the Azure portal.

[Create a scale set in the Azure portal](#)

Scaling in Service Fabric

3/5/2021 • 15 minutes to read • [Edit Online](#)

Azure Service Fabric makes it easy to build scalable applications by managing the services, partitions, and replicas on the nodes of a cluster. Running many workloads on the same hardware enables maximum resource utilization, but also provides flexibility in terms of how you choose to scale your workloads. This Channel 9 video describes how you can build scalable microservices applications:

Scaling in Service Fabric is accomplished several different ways:

1. Scaling by creating or removing stateless service instances
2. Scaling by creating or removing new named services
3. Scaling by creating or removing new named application instances
4. Scaling by using partitioned services
5. Scaling by adding and removing nodes from the cluster
6. Scaling by using Cluster Resource Manager metrics

Scaling by creating or removing stateless service instances

One of the simplest ways to scale within Service Fabric works with stateless services. When you create a stateless service, you get a chance to define an `InstanceCount`. `InstanceCount` defines how many running copies of that service's code are created when the service starts up. Let's say, for example, that there are 100 nodes in the cluster. Let's also say that a service is created with an `InstanceCount` of 10. During runtime, those 10 running copies of the code could all become too busy (or could be not busy enough). One way to scale that workload is to change the number of instances. For example, some piece of monitoring or management code can change the existing number of instances to 50, or to 5, depending on whether the workload needs to scale in or out based on the load.

C#:

```
StatelessServiceUpdateDescription updateDescription = new StatelessServiceUpdateDescription();
updateDescription.InstanceCount = 50;
await fabricClient.ServiceManager.UpdateServiceAsync(new Uri("fabric:/app/service"), updateDescription);
```

Powershell:

```
Update-ServiceFabricService -Stateless -ServiceName $serviceName -InstanceCount 50
```

Using Dynamic Instance Count

Specifically for stateless services, Service Fabric offers an automatic way to change the instance count. This allows the service to scale dynamically with the number of nodes that are available. The way to opt into this behavior is to set the instance count = -1. `InstanceCount = -1` is an instruction to Service Fabric that says "Run this stateless service on every node." If the number of nodes changes, Service Fabric automatically changes the instance count to match, ensuring that the service is running on all valid nodes.

C#:

```
StatelessServiceDescription serviceDescription = new StatelessServiceDescription();
//Set other service properties necessary for creation...
serviceDescription.InstanceCount = -1;
await fc.ServiceManager.CreateServiceAsync(serviceDescription);
```

Powershell:

```
New-ServiceFabricService -ApplicationName $applicationName -ServiceName $serviceName -ServiceTypeName
$serviceName -Stateless -PartitionSchemeSingleton -InstanceCount "-1"
```

Scaling by creating or removing new named services

A named service instance is a specific instance of a service type (see [Service Fabric application life cycle](#)) within some named application instance in the cluster.

New named service instances can be created (or removed) as services become more or less busy. This allows requests to be spread across more service instances, usually allowing load on existing services to decrease. When creating services, the Service Fabric Cluster Resource Manager places the services in the cluster in a distributed fashion. The exact decisions are governed by the [metrics](#) in the cluster and other placement rules. Services can be created several different ways, but the most common are either through administrative actions like someone calling `New-ServiceFabricService`, or by code calling `CreateServiceAsync`. `CreateServiceAsync` can even be called from within other services running in the cluster.

Creating services dynamically can be used in all sorts of scenarios, and is a common pattern. For example, consider a stateful service that represents a particular workflow. Calls representing work are going to show up to this service, and this service is going to execute the steps to that workflow and record progress.

How would you make this particular service scale? The service could be multi-tenant in some form, and accept calls and kick off steps for many different instances of the same workflow all at once. However, that can make the code more complex, since now it has to worry about many different instances of the same workflow, all at different stages and from different customers. Also, handling multiple workflows at the same time doesn't solve the scale problem. This is because at some point this service will consume too many resources to fit on a particular machine. Many services not built for this pattern in the first place also experience difficulty due to some inherent bottleneck or slowdown in their code. These types of issues cause the service not to work as well when the number of concurrent workflows it is tracking gets larger.

A solution is to create an instance of this service for every different instance of the workflow you want to track. This is a great pattern and works whether the service is stateless or stateful. For this pattern to work, there's usually another service that acts as a "Workload Manager Service". The job of this service is to receive requests and to route those requests to other services. The manager can dynamically create an instance of the workload service when it receives the message, and then pass on requests to those services. The manager service could also receive callbacks when a given workflow service completes its job. When the manager receives these callbacks it could delete that instance of the workflow service, or leave it if more calls are expected.

Advanced versions of this type of manager can even create pools of the services that it manages. The pool helps ensure that when a new request comes in it doesn't have to wait for the service to spin up. Instead, the manager can just pick a workflow service that is not currently busy from the pool, or route randomly. Keeping a pool of services available makes handling new requests faster, since it is less likely that the request has to wait for a new service to be spun up. Creating new services is quick, but not free or instantaneous. The pool helps minimize the amount of time the request has to wait before being serviced. You'll often see this manager and pool pattern when response times matter the most. Queuing the request and creating the service in the background and *then* passing it on is also a popular manager pattern, as is creating and deleting services based on some tracking of the amount of work that service currently has pending.

Scaling by creating or removing new named application instances

Creating and deleting whole application instances is similar to the pattern of creating and deleting services. For this pattern, there's some manager service that is making the decision based on the requests that it is seeing and the information it is receiving from the other services inside the cluster.

When should creating a new named application instance be used instead of creating a new named service instances in some already existing application? There's a few cases:

- The new application instance is for a customer whose code needs to run under some particular identity or security settings.
 - Service Fabric allows defining different code packages to run under particular identities. In order to launch the same code package under different identities, the activations need to occur in different application instances. Consider a case where you have an existing customer's workloads deployed. These may be running under a particular identity so you can monitor and control their access to other resources, such as remote databases or other systems. In this case, when a new customer signs up, you probably don't want to activate their code in the same context (process space). While you could, this makes it harder for your service code to act within the context of a particular identity. You typically must have more security, isolation, and identity management code. Instead of using different named service instances within the same application instance and hence the same process space, you can use different named Service Fabric Application instances. This makes it easier to define different identity contexts.
- The new application instance also serves as a means of configuration
 - By default, all of the named service instances of a particular service type within an application instance will run in the same process on a given node. What this means is that while you can configure each service instance differently, doing so is complicated. Services must have some token they use to look up their config within a configuration package. Usually this is just the service's name. This works fine, but it couples the configuration to the names of the individual named service instances within that application instance. This can be confusing and hard to manage since configuration is normally a design time artifact with application instance specific values. Creating more services always means more application upgrades to change the information within the config packages or to deploy new ones so that the new services can look up their specific information. It's often easier to create a whole new named application instance. Then you can use the application parameters to set whatever configuration is necessary for the services. This way all of the services that are created within that named application instance can inherit particular configuration settings. For example, instead of having a single configuration file with the settings and customizations for every customer, such as secrets or per customer resource limits, you'd instead have a different application instance for each customer with these settings overridden.
- The new application serves as an upgrade boundary
 - Within Service Fabric, different named application instances serve as boundaries for upgrade. An upgrade of one named application instance will not impact the code that another named application instance is running. The different applications will end up running different versions of the same code on the same nodes. This can be a factor when you need to make a scaling decision because you can choose whether the new code should follow the same upgrades as another service or not. For example, say that a call arrives at the manager service that is responsible for scaling a particular customer's workloads by creating and deleting services dynamically. In this case however, the call is for a workload associated with a *new* customer. Most customers like being isolated from each other not just for the security and configuration reasons listed previously, but because it provides more flexibility in terms of running specific versions of the software and choosing when they get upgraded. You may also create a new application instance and create the service there simply to further partition the amount of your services that any one upgrade will touch. Separate application instances provide greater granularity when doing application upgrades, and also enable A/B testing and Blue/Green

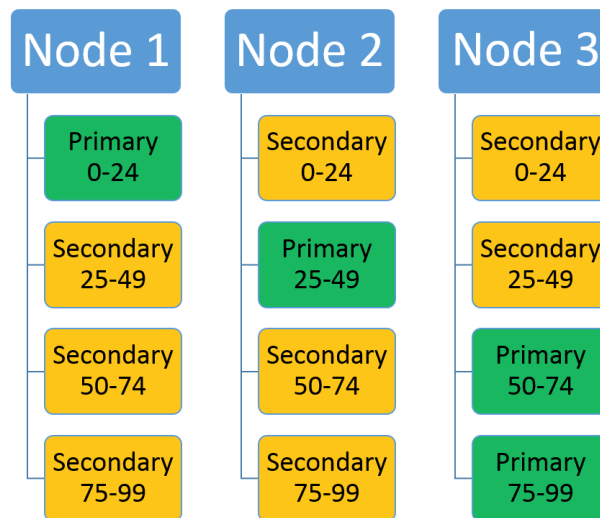
deployments.

- The existing application instance is full
 - In Service Fabric, [application capacity](#) is a concept that you can use to control the amount of resources available for particular application instances. For example, you may decide that a given service needs to have another instance created in order to scale. However, this application instance is out of capacity for a certain metric. If this particular customer or workload should still be granted more resources, then you could either increase the existing capacity for that application or create a new application.

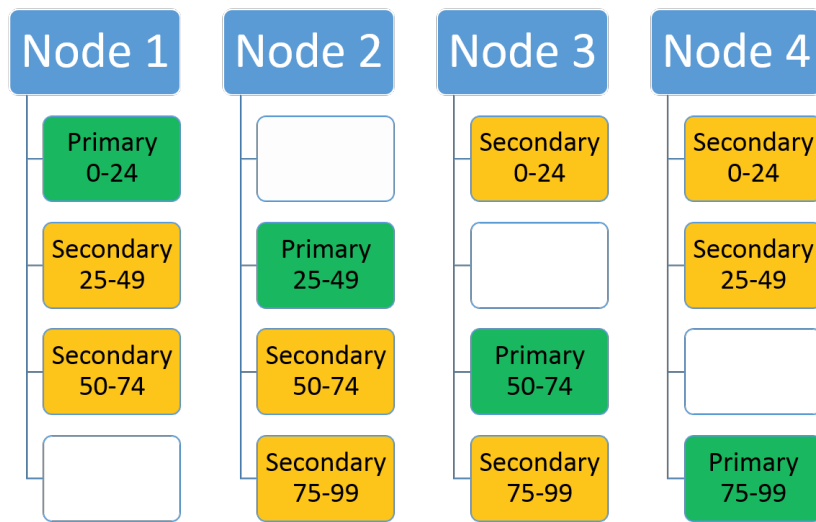
Scaling at the partition level

Service Fabric supports partitioning. Partitioning splits a service into several logical and physical sections, each of which operates independently. This is useful with stateful services, since no one set of replicas has to handle all the calls and manipulate all of the state at once. The [partitioning overview](#) provides information on the types of partitioning schemes that are supported. The replicas of each partition are spread across the nodes in a cluster, distributing that service's load and ensuring that neither the service as a whole or any partition has a single point of failure.

Consider a service that uses a ranged partitioning scheme with a low key of 0, a high key of 99, and a partition count of 4. In a three-node cluster, the service might be laid out with four replicas that share the resources on each node as shown here:



If you increase the number of nodes, Service Fabric will move some of the existing replicas there. For example, let's say the number of nodes increases to four and the replicas get redistributed. Now the service now has three replicas running on each node, each belonging to different partitions. This allows better resource utilization since the new node isn't cold. Typically, it also improves performance as each service has more resources available to it.



Scaling by using the Service Fabric Cluster Resource Manager and metrics

Metrics are how services express their resource consumption to Service Fabric. Using metrics gives the Cluster Resource Manager an opportunity to reorganize and optimize the layout of the cluster. For example, there may be plenty of resources in the cluster, but they might not be allocated to the services that are currently doing work. Using metrics allows the Cluster Resource Manager to reorganize the cluster to ensure that services have access to the available resources.

Scaling by adding and removing nodes from the cluster

Another option for scaling with Service Fabric is to change the size of the cluster. Changing the size of the cluster means adding or removing nodes for one or more of the node types in the cluster. For example, consider a case where all of the nodes in the cluster are hot. This means that the cluster's resources are almost all consumed. In this case, adding more nodes to the cluster is the best way to scale. Once the new nodes join the cluster the Service Fabric Cluster Resource Manager moves services to them, resulting in less total load on the existing nodes. For stateless services with instance count = -1, more service instances are automatically created. This allows some calls to move from the existing nodes to the new nodes.

For more information, see [cluster scaling](#).

Choosing a platform

Due to implementation differences between operating systems, choosing to use Service Fabric with Windows or Linux can be a vital part of scaling your application. One potential barrier is how staged logging is performed. Service Fabric on Windows uses a kernel driver for a one-per-machine log, shared between stateful service replicas. This log weighs in at about 8 GB. Linux, on the other hand, uses a 256 MB staging log for each replica, making it less ideal for applications that want to maximize the number of lightweight service replicas running on a given node. These differences in temporary storage requirements could potentially inform the desired platform for Service Fabric cluster deployment.

Putting it all together

Let's take all the ideas that we've discussed here and talk through an example. Consider the following service: you are trying to build a service that acts as an address book, holding on to names and contact information.

Right up front you have a bunch of questions related to scale: How many users are you going to have? How many contacts will each user store? Trying to figure this all out when you are standing up your service for the first time is difficult. Let's say you were going to go with a single static service with a specific partition count. The

consequences of picking the wrong partition count could cause you to have scale issues later. Similarly, even if you pick the right count you might not have all the information you need. For example, you also have to decide the size of the cluster up front, both in terms of the number of nodes and their sizes. It's usually hard to predict how many resources a service is going to consume over its lifetime. It can also be hard to know ahead of time the traffic pattern that the service actually sees. For example, maybe people add and remove their contacts only first thing in the morning, or maybe it's distributed evenly over the course of the day. Based on this you might need to scale out and in dynamically. Maybe you can learn to predict when you're going to need to scale out and in, but either way you're probably going to need to react to changing resource consumption by your service. This can involve changing the size of the cluster in order to provide more resources when reorganizing use of existing resources isn't enough.

But why even try to pick a single partition scheme out for all users? Why limit yourself to one service and one static cluster? The real situation is usually more dynamic.

When building for scale, consider the following dynamic pattern. You may need to adapt it to your situation:

1. Instead of trying to pick a partitioning scheme for everyone up front, build a "manager service".
2. The job of the manager service is to look at customer information when they sign up for your service. Then depending on that information the manager service creates an instance of your *actual* contact-storage service *just for that customer*. If they require particular configuration, isolation, or upgrades, you can also decide to spin up an Application instance for this customer.

This dynamic creation pattern many benefits:

- You're not trying to guess the correct partition count for all users up front or build a single service that is infinitely scalable all on its own.
- Different users don't have to have the same partition count, replica count, placement constraints, metrics, default loads, service names, dns settings, or any of the other properties specified at the service or application level.
- You gain additional data segmentation. Each customer has their own copy of the service
 - Each customer service can be configured differently and granted more or fewer resources, with more or fewer partitions or replicas as necessary based on their expected scale.
 - For example, say the customer paid for the "Gold" tier - they could get more replicas or greater partition count, and potentially resources dedicated to their services via metrics and application capacities.
 - Or say they provided information indicating the number of contacts they needed was "Small" - they would get only a few partitions, or could even be put into a shared service pool with other customers.
- You're not running a bunch of service instances or replicas while you're waiting for customers to show up
- If a customer ever leaves, then removing their information from your service is as simple as having the manager delete that service or application that it created.

Next steps

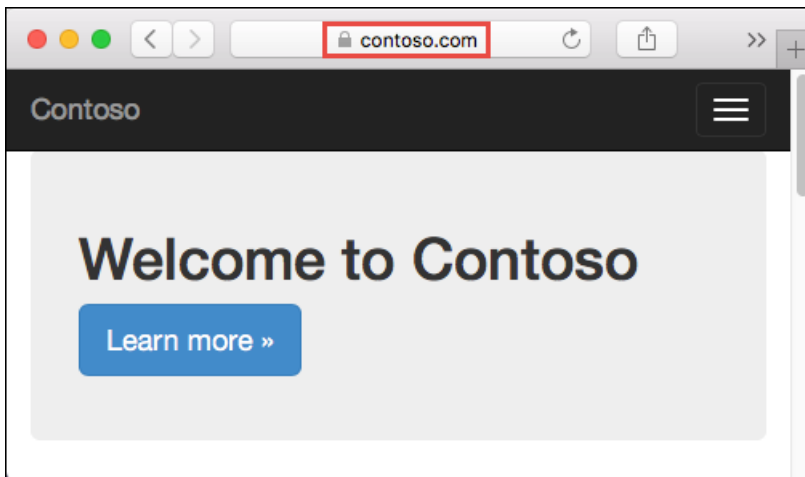
For more information on Service Fabric concepts, see the following articles:

- [Availability of Service Fabric services](#)
- [Partitioning Service Fabric services](#)

Secure a custom DNS name with a TLS/SSL binding in Azure App Service

6/24/2021 • 8 minutes to read • [Edit Online](#)

This article shows you how to secure the [custom domain](#) in your [App Service app](#) or [function app](#) by creating a certificate binding. When you're finished, you can access your App Service app at the `https://` endpoint for your custom DNS name (for example, `https://www.contoso.com`).



Securing a [custom domain](#) with a certificate involves two steps:

- [Add a private certificate to App Service](#) that satisfies all the [private certificate requirements](#).
- Create a TLS binding to the corresponding custom domain. This second step is covered by this article.

In this tutorial, you learn how to:

- Upgrade your app's pricing tier
- Secure a custom domain with a certificate
- Enforce HTTPS
- Enforce TLS 1.1/1.2
- Automate TLS management with scripts

Prerequisites

To follow this how-to guide:

- [Create an App Service app](#)
- [Map a domain name to your app](#) or [buy and configure it in Azure](#)
- [Add a private certificate to your app](#)

NOTE

The easiest way to add a private certificate is to [create a free App Service managed certificate](#).

Prepare your web app

To create custom TLS/SSL bindings or enable client certificates for your App Service app, your [App Service plan](#)

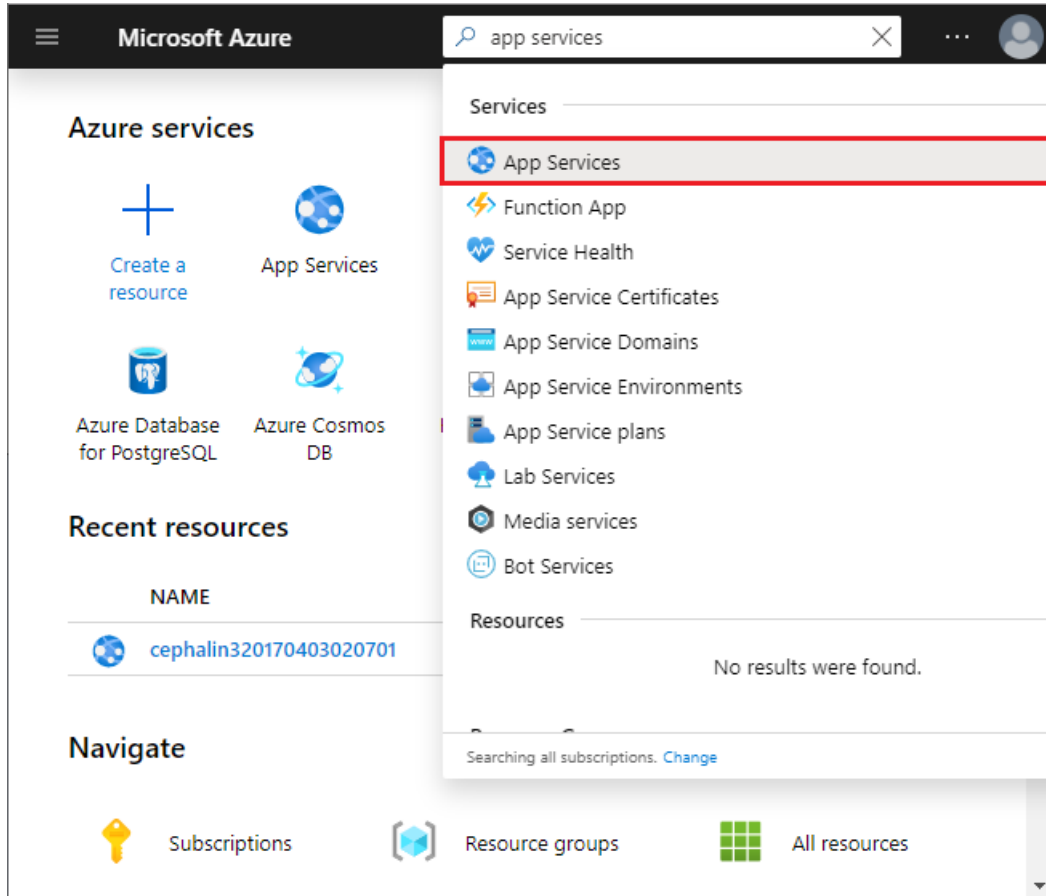
must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

Sign in to Azure

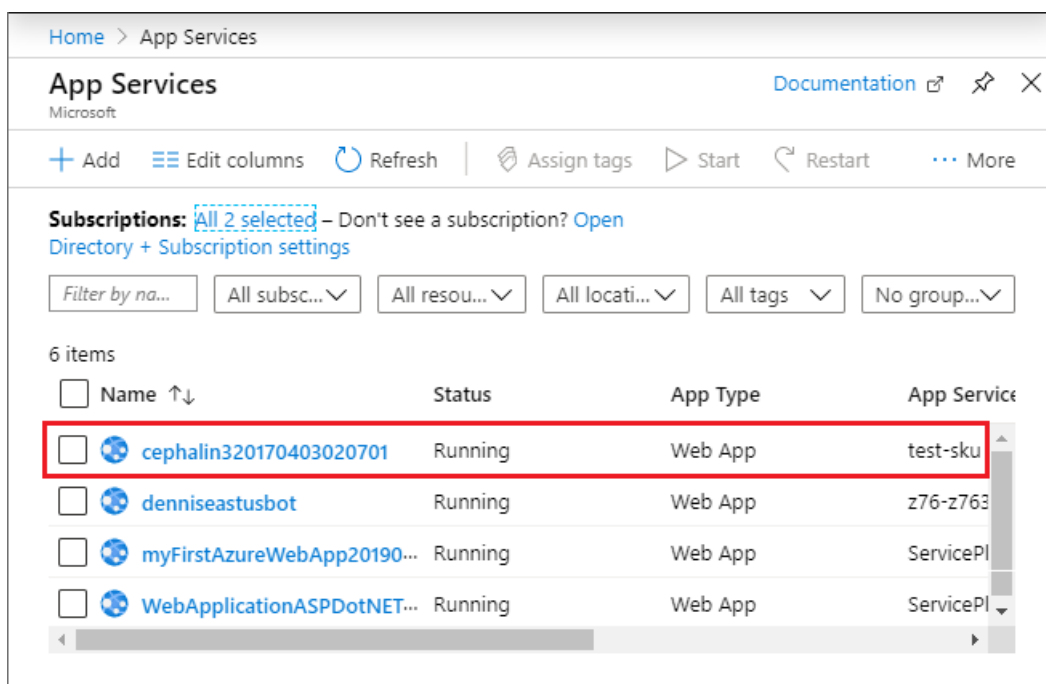
Open the [Azure portal](#).

Navigate to your web app

Search for and select **App Services**.



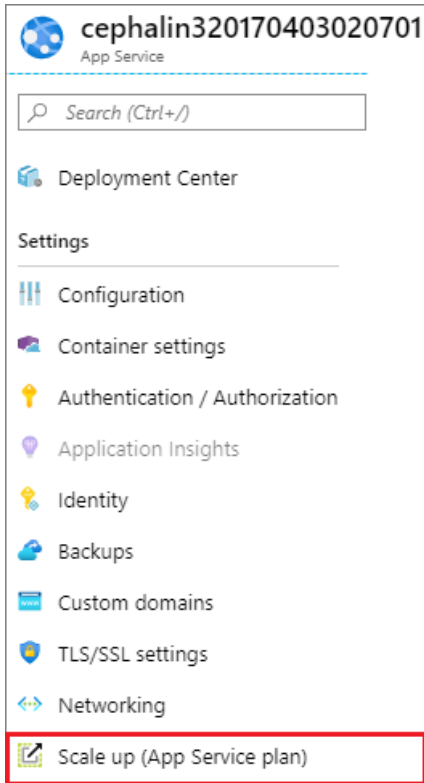
On the **App Services** page, select the name of your web app.








You have landed on the management page of your web app.

Check the pricing tier

In the left-hand navigation of your web app page, scroll to the **Settings** section and select **Scale up (App Service plan)**.



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


 Dev / Test For less demanding workloads	 Production For most production workloads	 Isolated Advanced networking and scale
<h3>Recommended pricing tiers</h3>		
<div style="border: 2px solid blue; padding: 10px;"> <p>F1</p> <p>Shared infrastructure 1 GB memory 60 minutes/day compute Free</p> </div>	<div style="padding: 10px;"> <p>D1</p> <p>Shared infrastructure 1 GB memory 240 minutes/day compute <price>/Month (Estimated)</p> </div>	
<div style="padding: 10px;"> <p>B1</p> <p>100 total ACU 1.75 GB memory A-Series compute equivalent <price>/Month (Estimated)</p> </div>		
▼ See additional options		
<h3>Included hardware</h3>		
<p>Every instance of your App Service plan will include the following hardware configuration:</p>		
<div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;">  <p>Azure Compute Units (ACU) Dedicated compute resources used to run applications deployed in the App...</p> </div>		
<div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;">  <p>Memory Memory available to run applications</p> </div>		

Custom SSL is not supported in the F1 or D1 tier. If you need to scale up, follow the steps in the next section. Otherwise, close the **Scale up** page and skip the [Scale up your App Service plan](#) section.


Scale up your App Service plan

Select any of the non-free tiers (B1, B2, B3, or any tier in the **Production** category). For additional options, click **See additional options**.


Click **Apply**.



Dev / Test
For less demanding workloads



Production
For most production workloads



Isolated
Advanced networking and scale

Recommended pricing tiers

F1

Shared infrastructure
1 GB memory
60 minutes/day compute
Free

D1

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


B1

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


[See additional options](#)

Included features

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




Custom domains / SSL
Configure and purchase custom domains with SNI SSL bindings




Manual scale
Up to 3 instances. Subject to availability.

Included hardware


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



Azure Compute Units (ACU)
Dedicated compute resources used to run applications deployed in the App...



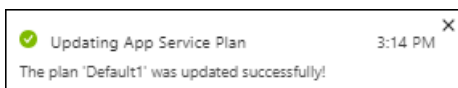
Memory
Memory per instance available to run applications deployed and running in...



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

Apply

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



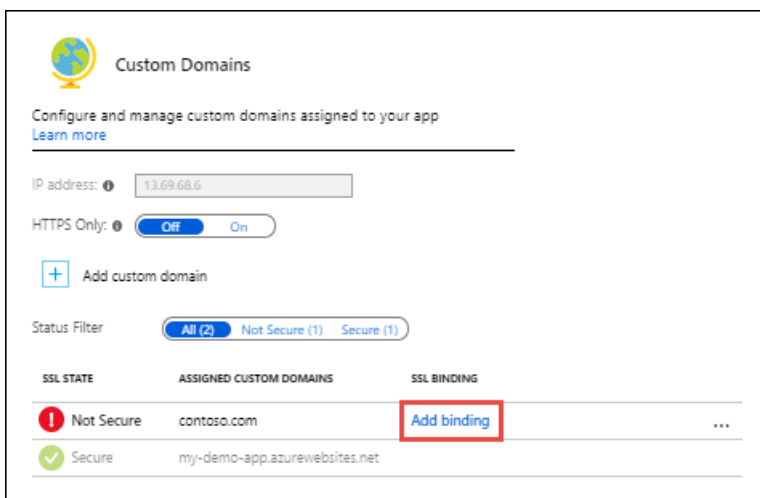
Secure a custom domain

Do the following steps:

In the [Azure portal](#), from the left menu, select **App Services** > <app-name>.

From the left navigation of your app, start the **TLS/SSL Binding** dialog by:

- Selecting **Custom domains > Add binding**
- Selecting **TLS/SSL settings > Add TLS/SSL binding**



In **Custom Domain**, select the custom domain you want to add a binding for.

If your app already has a certificate for the selected custom domain, go to [Create binding](#) directly. Otherwise, keep going.

Add a certificate for custom domain

If your app has no certificate for the selected custom domain, then you have two options:

- **Upload PFX Certificate** - Follow the workflow at [Upload a private certificate](#), then select this option here.
- **Import App Service Certificate** - Follow the workflow at [Import an App Service certificate](#), then select this option here.

NOTE

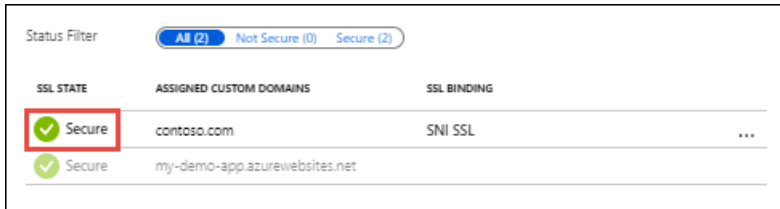
You can also [Create a free certificate](#) or [Import a Key Vault certificate](#), but you must do it separately and then return to the **TLS/SSL Binding** dialog.

Create binding

Use the following table to help you configure the TLS binding in the **TLS/SSL Binding** dialog, then click **Add Binding**.

SETTING	DESCRIPTION
Custom domain	The domain name to add the TLS/SSL binding for.
Private Certificate Thumbprint	The certificate to bind.
TLS/SSL Type	<ul style="list-style-type: none"> • SNI SSL - Multiple SNI SSL bindings may be added. This option allows multiple TLS/SSL certificates to secure multiple domains on the same IP address. Most modern browsers (including Internet Explorer, Chrome, Firefox, and Opera) support SNI (for more information, see Server Name Indication). • IP SSL - Only one IP SSL binding may be added. This option allows only one TLS/SSL certificate to secure a dedicated public IP address. After you configure the binding, follow the steps in Remap records for IP SSL. IP SSL is supported only in Standard tier or above.

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



SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
Secure	contoso.com	SNI SSL
Secure	my-demo-app.azurewebsites.net	SNI SSL

NOTE

A **Secure** state in the **Custom domains** means that it is secured with a certificate, but App Service doesn't check if the certificate is self-signed or expired, for example, which can also cause browsers to show an error or warning.

Remap records for IP SSL

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

There are two changes you need to make, potentially:

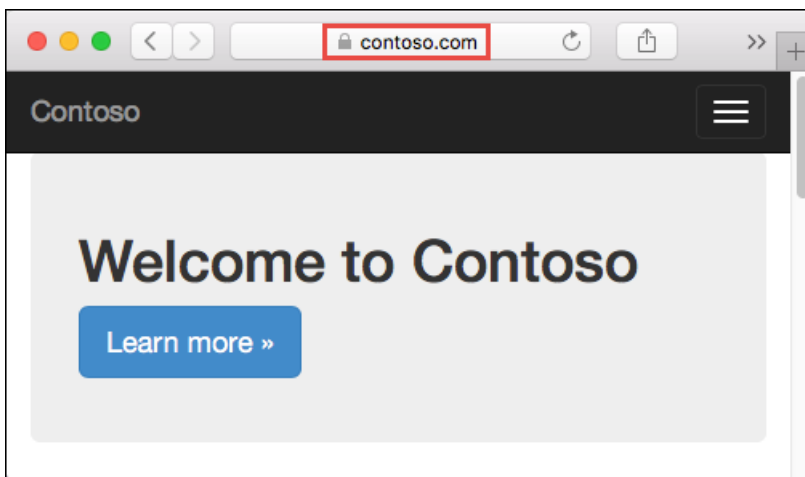
- By default, your app uses a shared public IP address. When you bind a certificate with IP SSL, App Service creates a new, dedicated IP address for your app. If you mapped an A record to your app, update your domain registry with this new, dedicated IP address.

Your app's **Custom domain** page is updated with the new, dedicated IP address. [Copy this IP address](#), then [remap the A record](#) to this new IP address.

- If you have an SNI SSL binding to `<app-name>.azurewebsites.net`, [remap any CNAME mapping](#) to point to `sni.<app-name>.azurewebsites.net` instead (add the `sni` prefix).

Test HTTPS

In various browsers, browse to `https://<your.custom.domain>` to verify that it serves up your app.



Your application code can inspect the protocol via the "x-appservice-proto" header. The header will have a value of `http` or `https`.

NOTE

If your app gives you certificate validation errors, you're probably using a self-signed certificate.

If that's not the case, you may have left out intermediate certificates when you export your certificate to the PFX file.

Prevent IP changes

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

1. Upload the new certificate.
2. Bind the new certificate to the custom domain you want without deleting the old one. This action replaces the binding instead of removing the old one.
3. Delete the old certificate.

Enforce HTTPS

By default, anyone can still access your app using HTTP. You can redirect all HTTP requests to the HTTPS port.

In your app page, in the left navigation, select **TLS/SSL settings**. Then, in **HTTPS Only**, select **On**.

The screenshot shows the Azure App Service management console for 'TLS/SSL settings'. The left-hand navigation menu is visible, with 'TLS/SSL settings' highlighted. The main content area is divided into sections: 'Bindings' (with sub-sections for Private Key Certificates (.pfx) and Public Key Certificates (.cer)), 'Protocol Settings' (containing a 'HTTPS Only' toggle set to 'On' and a 'Minimum TLS Version' dropdown set to '1.0'), and 'TLS/SSL bindings' (showing 'No TLS/SSL bindings configured for the app.').

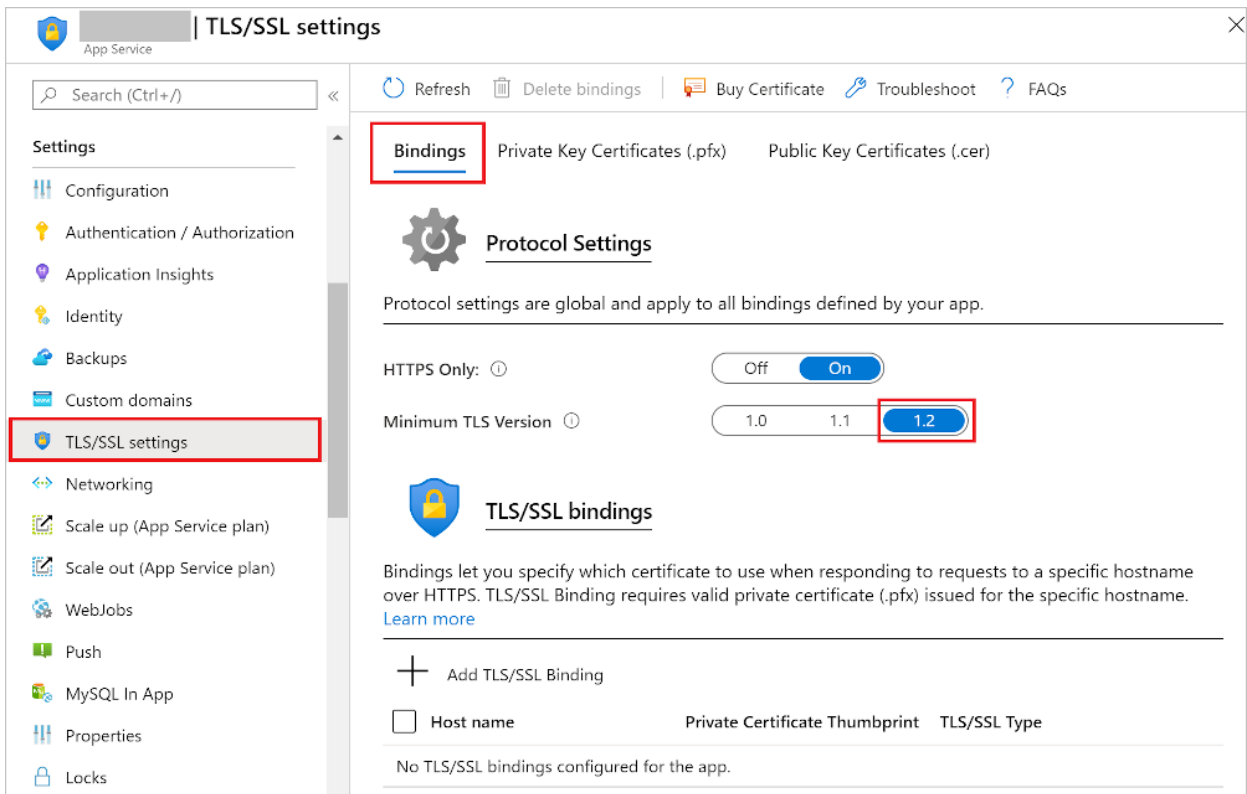
When the operation is complete, navigate to any of the HTTP URLs that point to your app. For example:

- `http://<app_name>.azurewebsites.net`
- `http://contoso.com`
- `http://www.contoso.com`

Enforce TLS versions

Your app allows **TLS 1.2** by default, which is the recommended TLS level by industry standards, such as **PCI DSS**. To enforce different TLS versions, follow these steps:

In your app page, in the left navigation, select **TLS/SSL settings**. Then, in **TLS version**, select the minimum TLS version you want. This setting controls the inbound calls only.



When the operation is complete, your app rejects all connections with lower TLS versions.

Handle TLS termination

In App Service, [TLS termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

Language specific configuration guides, such as the [Linux Node.js configuration](#) guide, shows you how to detect an HTTPS session in your application code.

Automate with scripts

Azure CLI

```

#!/bin/bash

fqdn=<replace-with-www.{yourdomain}>
pfxPath=<replace-with-path-to-your-.PFX-file>
pfxPassword=<replace-with-your=.PFX-password>
resourceGroup=myResourceGroup
webappName=mywebapp$RANDOM

# Create a resource group.
az group create --location westeurope --name $resourceGroup

# Create an App Service plan in Basic tier (minimum required by custom domains).
az appservice plan create --name $webappName --resource-group $resourceGroup --sku B1

# Create a web app.
az webapp create --name $webappName --resource-group $resourceGroup \
--plan $webappName

echo "Configure a CNAME record that maps $fqdn to $webappName.azurewebsites.net"
read -p "Press [Enter] key when ready ..."

# Before continuing, go to your DNS configuration UI for your custom domain and follow the
# instructions at https://aka.ms/appservicecustomdns to configure a CNAME record for the
# hostname "www" and point it your web app's default domain name.

# Map your prepared custom domain name to the web app.
az webapp config hostname add --webapp-name $webappName --resource-group $resourceGroup \
--hostname $fqdn

# Upload the SSL certificate and get the thumbprint.
thumbprint=$(az webapp config ssl upload --certificate-file $pfxPath \
--certificate-password $pfxPassword --name $webappName --resource-group $resourceGroup \
--query thumbprint --output tsv)

# Binds the uploaded SSL certificate to the web app.
az webapp config ssl bind --certificate-thumbprint $thumbprint --ssl-type SNI \
--name $webappName --resource-group $resourceGroup

echo "You can now browse to https://$fqdn"

```

PowerShell

```

$fqdn="<Replace with your custom domain name>"
$pfxPath="<Replace with path to your .PFX file>"
$pfxPassword="<Replace with your .PFX password>"
$webappname="mywebapp$(Get-Random)"
$location="West Europe"

# Create a resource group.
New-AzResourceGroup -Name $webappname -Location $location

# Create an App Service plan in Free tier.
New-AzAppServicePlan -Name $webappname -Location $location `
-ResourceGroupName $webappname -Tier Free

# Create a web app.
New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname `
-ResourceGroupName $webappname

Write-Host "Configure a CNAME record that maps $fqdn to $webappname.azurewebsites.net"
Read-Host "Press [Enter] key when ready ..."

# Before continuing, go to your DNS configuration UI for your custom domain and follow the
# instructions at https://aka.ms/appservicecustomdns to configure a CNAME record for the
# hostname "www" and point it your web app's default domain name.

# Upgrade App Service plan to Basic tier (minimum required by custom SSL certificates)
Set-AzAppServicePlan -Name $webappname -ResourceGroupName $webappname `
-Tier Basic

# Add a custom domain name to the web app.
Set-AzWebApp -Name $webappname -ResourceGroupName $webappname `
-HostNames @($fqdn,$webappname.azurewebsites.net)

# Upload and bind the SSL certificate to the web app.
New-AzWebAppSSLBinding -WebAppName $webappname -ResourceGroupName $webappname -Name $fqdn `
-CertificateFilePath $pfxPath -CertificatePassword $pfxPassword -SslState SniEnabled

```

More resources

- [Use a TLS/SSL certificate in your code in Azure App Service](#)
- [FAQ : App Service Certificates](#)

Back up your app in Azure

6/28/2021 • 8 minutes to read • [Edit Online](#)

The Backup and Restore feature in [Azure App Service](#) lets you easily create app backups manually or on a schedule. You can configure the backups to be retained up to an indefinite amount of time. You can restore the app to a snapshot of a previous state by overwriting the existing app or restoring to another app.

For information on restoring an app from backup, see [Restore an app in Azure](#).

What gets backed up

App Service can back up the following information to an Azure storage account and container that you have configured your app to use.

- App configuration
- File content
- Database connected to your app

The following database solutions are supported with backup feature:

- [SQL Database](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)
- [MySQL in-app](#)

NOTE

Each backup is a complete offline copy of your app, not an incremental update.

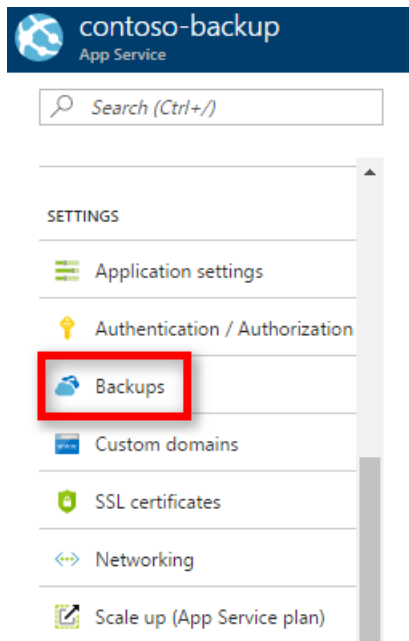
Requirements and restrictions

- The Backup and Restore feature requires the App Service plan to be in the **Standard**, **Premium**, or **Isolated** tier. For more information about scaling your App Service plan to use a higher tier, see [Scale up an app in Azure](#). **Premium** and **Isolated** tiers allow a greater number of daily back ups than **Standard** tier.
- You need an Azure storage account and container in the same subscription as the app that you want to back up. For more information on Azure storage accounts, see [Azure storage account overview](#).
- Backups can be up to 10 GB of app and database content. If the backup size exceeds this limit, you get an error.
- Backups of TLS enabled Azure Database for MySQL is not supported. If a backup is configured, you will encounter backup failures.
- Backups of TLS enabled Azure Database for PostgreSQL is not supported. If a backup is configured, you will encounter backup failures.
- In-app MySQL databases are automatically backed up without any configuration. If you make manually settings for in-app MySQL databases, such as adding connection strings, the backups may not work correctly.
- Using a firewall enabled storage account as the destination for your backups is not supported. If a backup is configured, you will encounter backup failures.
- Currently, you can't use the Backup and Restore feature with the Azure App Service VNet Integration feature.

- Currently, you can't use the Backup and Restore feature with Azure storage accounts that are configured to use Private Endpoint.

Create a manual backup

1. In the [Azure portal](#), navigate to your app's page, select **Backups**. The **Backups** page is displayed.



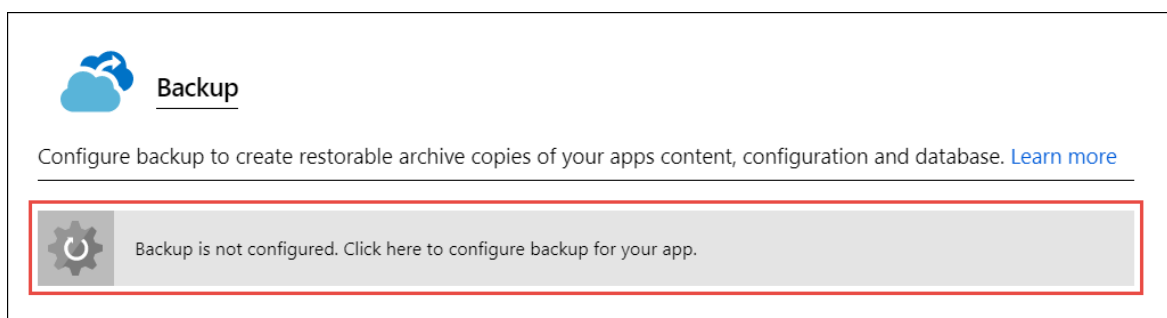
NOTE

If you see the following message, click it to upgrade your App Service plan before you can proceed with backups. For more information, see [Scale up an app in Azure](#).



Backup and Restore feature requires your App Service plan to be Standard or higher. [Click here to upgrade your App Service plan and access this feature.](#)

2. In the **Backup** page, select **Backup is not configured**. [Click here to configure backup for your app](#).



3. In the **Backup Configuration** page, click **Storage not configured** to configure a storage account.



Backup Storage

Select the target container to store your app backup.

Storage Settings

Storage not configured



4. Choose your backup destination by selecting a **Storage Account** and **Container**. The storage account must belong to the same subscription as the app you want to back up. If you wish, you can create a new storage account or a new container in the respective pages. When you're done, click **Select**.
5. In the **Backup Configuration** page that is still left open, you can configure **Backup Database**, then select the databases you want to include in the backups (SQL Database or MySQL), then click **OK**.



Backup Database

Select the databases to include with your backup. The backup database list is based on the app's configured connection strings. Note: The maximum size of content + database backup cannot exceed 10GB. if your database is large and growing, use Azure Backup for database backup instead.



INCLUDE IN BACKUP

CONNECTION STRING NAME

DATABASE TYPE

No supported connection strings of type SQL Database or MySQL found configured in app.

NOTE

For a database to appear in this list, its connection string must exist in the **Connection strings** section of the **Application settings** page for your app.

In-app MySQL databases are automatically backed up without any configuration. If you make settings for in-app MySQL databases manually, such as adding connection strings, the backups may not work correctly.

6. In the **Backup Configuration** page, click **Save**.
7. In the **Backups** page, click **Backup**.

Backup

Configure backup to create restorable archive copies of your apps content, configuration and database. [Learn more](#)

Backup configured, backup schedule is not configured, configure scheduled backup to automatically take backups.

Backup **Restore**

STATUS	BACKUP TIME	SIZE (MB)
InProgress	Wednesday, October 16, 2019, 6:20:10 PM	0

You see a progress message during the backup process.

Once the storage account and container is configured, you can initiate a manual backup at any time. Manual backups are retained indefinitely.

Configure automated backups

1. In the **Backup Configuration** page, set **Scheduled backup** to **On**.

Backup Schedule

Configure the schedule for your app backup.

Scheduled backup **On** Off

* Backup Every **Days** Hours

* Start backup schedule from

* Retention (Days)

Keep at least one backup **Yes** No

2. Configure the backup schedule as desired and select **OK**.

Configure Partial Backups

Sometimes you don't want to back up everything on your app. Here are a few examples:

- You [set up weekly backups](#) of your app that contains static content that never changes, such as old blog posts or images.
- Your app has over 10 GB of content (that's the max amount you can back up at a time).
- You don't want to back up the log files.

Partial backups allow you choose exactly which files you want to back up.

NOTE

Individual databases in the backup can be 4GB max but the total max size of the backup is 10GB







Exclude files from your backup

Suppose you have an app that contains log files and static images that have been backup once and are not going to change. In such cases, you can exclude those folders and files from being stored in your future backups. To exclude files and folders from your backups, create a `_backup.filter` file in the `D:\home\site\wwwroot` folder of your app. Specify the list of files and folders you want to exclude in this file.

You can access your files by navigating to `https://<app-name>.scm.azurewebsites.net/DebugConsole`. If prompted, sign in to your Azure account.

Identify the folders that you want to exclude from your backups. For example, you want to filter out the highlighted folder and files.

... / Images + | 6 items

	Name
	2013
	2014
	2015
	Products
	bkg.png
	brand.png

Create a file called `_backup.filter` and put the preceding list in the file, but remove `D:\home`. List one directory or file per line. So the content of the file should be:

```
\site\wwwroot\Images\brand.png
\site\wwwroot\Images\2014
\site\wwwroot\Images\2013
```

Upload `_backup.filter` file to the `D:\home\site\wwwroot\` directory of your site using [ftp](#) or any other method. If you wish, you can create the file directly using Kudu `DebugConsole` and insert the content there.

Run backups the same way you would normally do it, [manually](#) or [automatically](#). Now, any files and folders that are specified in `_backup.filter` is excluded from the future backups scheduled or manually initiated.

NOTE

You restore partial backups of your site the same way you would [restore a regular backup](#). The restore process does the right thing.

When a full backup is restored, all content on the site is replaced with whatever is in the backup. If a file is on the site, but not in the backup it gets deleted. But when a partial backup is restored, any content that is located in one of the restricted directories, or any restricted file, is left as is.

How backups are stored

After you have made one or more backups for your app, the backups are visible on the **Containers** page of your storage account, and your app. In the storage account, each backup consists of a `.zip` file that contains the backup data and an `.xml` file that contains a manifest of the `.zip` file contents. You can unzip and browse these files if you want to access your backups without actually performing an app restore.

The database backup for the app is stored in the root of the `.zip` file. For SQL Database, this is a BACPAC file (no file extension) and can be imported. To create a database in Azure SQL Database based on the BACPAC export, see [Import a BACPAC file to create a database in Azure SQL Database](#).

WARNING

Altering any of the files in your **websitebackups** container can cause the backup to become invalid and therefore non-restorable.

Troubleshooting

The **Backups** page shows you the status of each backup. If you click on a failed backup, you can get log details regarding the failure. Use the following table to help you troubleshoot your backup. If the failure isn't documented in the table, open a support ticket.

ERROR	FIX
Storage access failed.	Delete backup schedule and reconfigure it. Or, reconfigure the backup storage.
The website + database size exceeds the {0} GB limit for backups. Your content size is {1} GB.	Exclude some files from the backup, or remove the database portion of the backup and use externally offered backups instead.
Error occurred while connecting to the database {0} on server {1}: Authentication to host '{1}' for user '<username>' using method 'mysql_native_password' failed with message: Unknown database '<db-name>'	Update database connection string.
Cannot resolve {0}. {1} (CannotResolveStorageAccount)	Delete the backup schedule and reconfigure it.
Login failed for user '{0}'.	Update the database connection string.
Create Database copy of {0} ({1}) threw an exception. Could not create Database copy.	Use an administrative user in the connection string.

ERROR	FIX
The server principal "<name>" is not able to access the database "master" under the current security context. Cannot open database "master" requested by the login. The login failed. Login failed for user '<name>'.	Use an administrative user in the connection string.
A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server).	Check that the connection string is valid. Allow the app's outbound IPs in the database server settings.
Cannot open server "<name>" requested by the login. The login failed.	Check that the connection string is valid.
Missing mandatory parameters for valid Shared Access Signature.	Delete the backup schedule and reconfigure it.
SSL connection is required. Please specify SSL options and retry. when trying to connect.	Use the built-in backup feature in Azure MySQL or Azure Postgresql instead.

Automate with scripts

You can automate backup management with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

For samples, see:

- [Azure CLI samples](#)
- [Azure PowerShell samples](#)

Next Steps

For information on restoring an app from a backup, see [Restore an app in Azure](#).

An overview of Azure VM backup

4/19/2021 • 12 minutes to read • [Edit Online](#)

This article describes how the [Azure Backup service](#) backs up Azure virtual machines (VMs).

Azure Backup provides independent and isolated backups to guard against unintended destruction of the data on your VMs. Backups are stored in a Recovery Services vault with built-in management of recovery points. Configuration and scaling are simple, backups are optimized, and you can easily restore as needed.

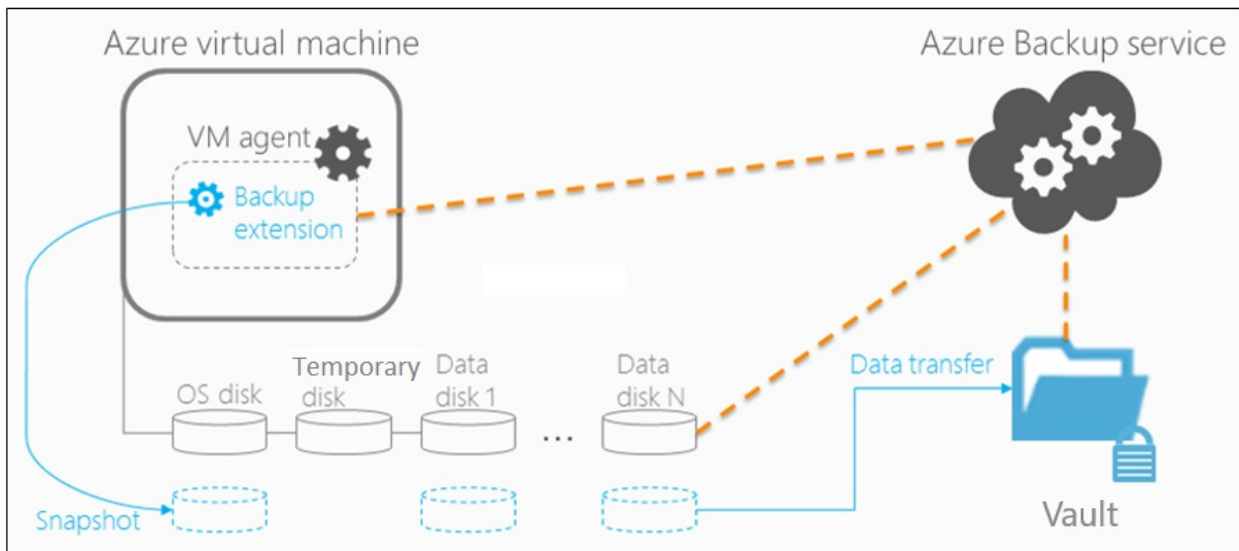
As part of the backup process, a [snapshot is taken](#), and the data is transferred to the Recovery Services vault with no impact on production workloads. The snapshot provides different levels of consistency, as described [here](#).

Azure Backup also has specialized offerings for database workloads like [SQL Server](#) and [SAP HANA](#) that are workload-aware, offer 15 minute RPO (recovery point objective), and allow backup and restore of individual databases.

Backup process

Here's how Azure Backup completes a backup for Azure VMs:

1. For Azure VMs that are selected for backup, Azure Backup starts a backup job according to the backup schedule you specify.
2. During the first backup, a backup extension is installed on the VM if the VM is running.
 - For Windows VMs, the [VMSnapshot extension](#) is installed.
 - For Linux VMs, the [VMSnapshotLinux extension](#) is installed.
3. For Windows VMs that are running, Backup coordinates with Windows Volume Shadow Copy Service (VSS) to take an app-consistent snapshot of the VM.
 - By default, Backup takes full VSS backups.
 - If Backup can't take an app-consistent snapshot, then it takes a file-consistent snapshot of the underlying storage (because no application writes occur while the VM is stopped).
4. For Linux VMs, Backup takes a file-consistent backup. For app-consistent snapshots, you need to manually customize pre/post scripts.
5. After Backup takes the snapshot, it transfers the data to the vault.
 - The backup is optimized by backing up each VM disk in parallel.
 - For each disk that's being backed up, Azure Backup reads the blocks on the disk and identifies and transfers only the data blocks that changed (the delta) since the previous backup.
 - Snapshot data might not be immediately copied to the vault. It might take some hours at peak times. Total backup time for a VM will be less than 24 hours for daily backup policies.
6. Changes made to a Windows VM after Azure Backup is enabled on it are:
 - Microsoft Visual C++ 2013 Redistributable(x64) - 12.0.40660 is installed in the VM
 - Startup type of Volume Shadow Copy service (VSS) changed to automatic from manual
 - IaaSVmProvider Windows service is added
7. When the data transfer is complete, the snapshot is removed, and a recovery point is created.



Encryption of Azure VM backups

When you back up Azure VMs with Azure Backup, VMs are encrypted at rest with Storage Service Encryption (SSE). Azure Backup can also back up Azure VMs that are encrypted by using Azure Disk Encryption.

ENCRYPTION	DETAILS	SUPPORT
SSE	<p>With SSE, Azure Storage provides encryption at rest by automatically encrypting data before storing it. Azure Storage also decrypts data before retrieving it. Azure Backup supports backups of VMs with two types of Storage Service Encryption:</p> <ul style="list-style-type: none"> • SSE with platform-managed keys: This encryption is by default for all disks in your VMs. See more here. • SSE with customer-managed keys. With CMK, you manage the keys used to encrypt the disks. See more here. 	<p>Azure Backup uses SSE for at-rest encryption of Azure VMs.</p>
Azure Disk Encryption	<p>Azure Disk Encryption encrypts both OS and data disks for Azure VMs.</p> <p>Azure Disk Encryption integrates with BitLocker encryption keys (BEKs), which are safeguarded in a key vault as secrets. Azure Disk Encryption also integrates with Azure Key Vault key encryption keys (KEKs).</p>	<p>Azure Backup supports backup of managed and unmanaged Azure VMs encrypted with BEKs only, or with BEKs together with KEKs.</p> <p>Both BEKs and KEKs are backed up and encrypted.</p> <p>Because KEKs and BEKs are backed up, users with the necessary permissions can restore keys and secrets back to the key vault if needed. These users can also recover the encrypted VM.</p> <p>Encrypted keys and secrets can't be read by unauthorized users or by Azure.</p>

For managed and unmanaged Azure VMs, Backup supports both VMs encrypted with BEKs only or VMs encrypted with BEKs together with KEKs.

The backed-up BEKs (secrets) and KEKs (keys) are encrypted. They can be read and used only when they're

restored back to the key vault by authorized users. Neither unauthorized users, or Azure, can read or use backed-up keys or secrets.

BEKs are also backed up. So, if the BEKs are lost, authorized users can restore the BEKs to the key vault and recover the encrypted VMs. Only users with the necessary level of permissions can back up and restore encrypted VMs or keys and secrets.

Snapshot creation

Azure Backup takes snapshots according to the backup schedule.

- **Windows VMs:** For Windows VMs, the Backup service coordinates with VSS to take an app-consistent snapshot of the VM disks. By default, Azure Backup takes a full VSS backup (it truncates the logs of application such as SQL Server at the time of backup to get application level consistent backup). If you're using a SQL Server database on Azure VM backup, then you can modify the setting to take a VSS Copy backup (to preserve logs). For more information, see [this article](#).
- **Linux VMs:** To take app-consistent snapshots of Linux VMs, use the Linux pre-script and post-script framework to write your own custom scripts to ensure consistency.
 - Azure Backup invokes only the pre/post scripts written by you.
 - If the pre-scripts and post-scripts execute successfully, Azure Backup marks the recovery point as application-consistent. However, when you're using custom scripts, you're ultimately responsible for the application consistency.
 - [Learn more](#) about how to configure scripts.

Snapshot consistency

The following table explains the different types of snapshot consistency:

SNAPSHOT	DETAILS	RECOVERY	CONSIDERATION
Application-consistent	App-consistent backups capture memory content and pending I/O operations. App-consistent snapshots use a VSS writer (or pre/post scripts for Linux) to ensure the consistency of the app data before a backup occurs.	When you're recovering a VM with an app-consistent snapshot, the VM boots up. There's no data corruption or loss. The apps start in a consistent state.	Windows: All VSS writers succeeded Linux: Pre/post scripts are configured and succeeded
File-system consistent	File-system consistent backups provide consistency by taking a snapshot of all files at the same time.	When you're recovering a VM with a file-system consistent snapshot, the VM boots up. There's no data corruption or loss. Apps need to implement their own "fix-up" mechanism to make sure that restored data is consistent.	Windows: Some VSS writers failed Linux: Default (if pre/post scripts aren't configured or failed)

SNAPSHOT	DETAILS	RECOVERY	CONSIDERATION
Crash-consistent	Crash-consistent snapshots typically occur if an Azure VM shuts down at the time of backup. Only the data that already exists on the disk at the time of backup is captured and backed up.	Starts with the VM boot process followed by a disk check to fix corruption errors. Any in-memory data or write operations that weren't transferred to disk before the crash are lost. Apps implement their own data verification. For example, a database app can use its transaction log for verification. If the transaction log has entries that aren't in the database, the database software rolls transactions back until the data is consistent.	VM is in shutdown (stopped/ deallocated) state.

NOTE

If the provisioning state is **succeeded**, Azure Backup takes file-system consistent backups. If the provisioning state is **unavailable** or **failed**, crash-consistent backups are taken. If the provisioning state is **creating** or **deleting**, that means Azure Backup is retrying the operations.

Backup and restore considerations

CONSIDERATION	DETAILS
Disk	Backup of VM disks is parallel. For example, if a VM has four disks, the Backup service attempts to back up all four disks in parallel. Backup is incremental (only changed data).
Scheduling	To reduce backup traffic, back up different VMs at different times of the day and make sure the times don't overlap. Backing up VMs at the same time causes traffic jams.
Preparing backups	Keep in mind the time needed to prepare the backup. The preparation time includes installing or updating the backup extension and triggering a snapshot according to the backup schedule.

CONSIDERATION	DETAILS
Data transfer	<p>Consider the time needed for Azure Backup to identify the incremental changes from the previous backup.</p> <p>In an incremental backup, Azure Backup determines the changes by calculating the checksum of the block. If a block is changed, it's marked for transfer to the vault. The service analyzes the identified blocks to attempt to further minimize the amount of data to transfer. After evaluating all the changed blocks, Azure Backup transfers the changes to the vault.</p> <p>There might be a lag between taking the snapshot and copying it to vault. At peak times, it can take up to eight hours for the snapshots to be transferred to the vault. The backup time for a VM will be less than 24 hours for the daily backup.</p>
Initial backup	<p>Although the total backup time for incremental backups is less than 24 hours, that might not be the case for the first backup. The time needed for the initial backup will depend on the size of the data and when the backup is processed.</p>
Restore queue	<p>Azure Backup processes restore jobs from multiple storage accounts at the same time, and it puts restore requests in a queue.</p>
Restore copy	<p>During the restore process, data is copied from the vault to the storage account.</p> <p>The total restore time depends on the I/O operations per second (IOPS) and the throughput of the storage account.</p> <p>To reduce the copy time, select a storage account that isn't loaded with other application writes and reads.</p>

Backup performance

These common scenarios can affect the total backup time:

- **Adding a new disk to a protected Azure VM:** If a VM is undergoing incremental backup and a new disk is added, the backup time will increase. The total backup time might last more than 24 hours because of initial replication of the new disk, along with delta replication of existing disks.
- **Fragmented disks:** Backup operations are faster when disk changes are contiguous. If changes are spread out and fragmented across a disk, backup will be slower.
- **Disk churn:** If protected disks that are undergoing incremental backup have a daily churn of more than 200 GB, backup can take a long time (more than eight hours) to complete.
- **Backup versions:** The latest version of Backup (known as the Instant Restore version) uses a more optimized process than checksum comparison for identifying changes. But if you're using Instant Restore and have deleted a backup snapshot, the backup switches to checksum comparison. In this case, the backup operation will exceed 24 hours (or fail).

Restore performance

These common scenarios can affect the total restore time:

- The total restore time depends on the Input/output operations per second (IOPS) and the throughput of the storage account.
- The total restore time can be affected if the target storage account is loaded with other application read and

write operations. To improve restore operation, select a storage account that isn't loaded with other application data.

Best practices

When you're configuring VM backups, we suggest following these practices:

- Modify the default schedule times that are set in a policy. For example, if the default time in the policy is 12:00 AM, increment the timing by several minutes so that resources are optimally used.
- If you're restoring VMs from a single vault, we highly recommend that you use different [general-purpose v2 storage accounts](#) to ensure that the target storage account doesn't get throttled. For example, each VM must have a different storage account. For example, if 10 VMs are restored, use 10 different storage accounts.
- For backup of VMs that are using premium storage with Instant Restore, we recommend allocating *50%* free space of the total allocated storage space, which is required **only** for the first backup. The 50% free space isn't a requirement for backups after the first backup is complete
- The limit on the number of disks per storage account is relative to how heavily the disks are being accessed by applications that are running on an infrastructure as a service (IaaS) VM. As a general practice, if 5 to 10 disks or more are present on a single storage account, balance the load by moving some disks to separate storage accounts.
- To restore VMs with managed disks using PowerShell, provide the additional parameter *TargetResourceGroupName* to specify the resource group to which managed disks will be restored, [Learn more here](#).

Backup costs

Azure VMs backed up with Azure Backup are subject to [Azure Backup pricing](#).

Billing doesn't start until the first successful backup finishes. At this point, the billing for both storage and protected VMs begins. Billing continues as long as any backup data for the VM is stored in a vault. If you stop protection for a VM, but backup data for the VM exists in a vault, billing continues.

Billing for a specified VM stops only if the protection is stopped and all backup data is deleted. When protection stops and there are no active backup jobs, the size of the last successful VM backup becomes the protected instance size used for the monthly bill.

The protected-instance size calculation is based on the *actual* size of the VM. The VM's size is the sum of all the data in the VM, excluding the temporary storage. Pricing is based on the actual data that's stored on the data disks, not on the maximum supported size for each data disk that's attached to the VM.

Similarly, the backup storage bill is based on the amount of data that's stored in Azure Backup, which is the sum of the actual data in each recovery point.

For example, take an A2-Standard-sized VM that has two additional data disks with a maximum size of 32 TB each. The following table shows the actual data stored on each of these disks:

DISK	MAX SIZE	ACTUAL DATA PRESENT
OS disk	32 TB	17 GB
Local/temporary disk	135 GB	5 GB (not included for backup)
Data disk 1	32 TB	30 GB
Data disk 2	32 TB	0 GB

The actual size of the VM in this case is $17\text{ GB} + 30\text{ GB} + 0\text{ GB} = 47\text{ GB}$. This protected-instance size (47 GB) becomes the basis for the monthly bill. As the amount of data in the VM grows, the protected-instance size used for billing changes to match.

Next steps

- [Prepare for Azure VM backup.](#)

Enable diagnostics logging for apps in Azure App Service

4/27/2021 • 9 minutes to read • [Edit Online](#)

Overview

Azure provides built-in diagnostics to assist with debugging an [App Service app](#). In this article, you learn how to enable diagnostic logging and add instrumentation to your application, as well as how to access the information logged by Azure.

This article uses the [Azure portal](#) and Azure CLI to work with diagnostic logs. For information on working with diagnostic logs using Visual Studio, see [Troubleshooting Azure in Visual Studio](#).

NOTE

In addition to the logging instructions in this article, there's new, integrated logging capability with Azure Monitoring. You'll find more on this capability in the [Send logs to Azure Monitor \(preview\)](#) section.

TYPE	PLATFORM	LOCATION	DESCRIPTION
Application logging	Windows, Linux	App Service file system and/or Azure Storage blobs	Logs messages generated by your application code. The messages can be generated by the web framework you choose, or from your application code directly using the standard logging pattern of your language. Each message is assigned one of the following categories: Critical, Error, Warning, Info, Debug, and Trace . You can select how verbose you want the logging to be by setting the severity level when you enable application logging.
Web server logging	Windows	App Service file system or Azure Storage blobs	Raw HTTP request data in the W3C extended log file format . Each log message includes data such as the HTTP method, resource URI, client IP, client port, user agent, response code, and so on.

TYPE	PLATFORM	LOCATION	DESCRIPTION
Detailed Error Messages	Windows	App Service file system	Copies of the <i>.htm</i> error pages that would have been sent to the client browser. For security reasons, detailed error pages shouldn't be sent to clients in production, but App Service can save the error page each time an application error occurs that has HTTP code 400 or greater. The page may contain information that can help determine why the server returns the error code.
Failed request tracing	Windows	App Service file system	Detailed tracing information on failed requests, including a trace of the IIS components used to process the request and the time taken in each component. It's useful if you want to improve site performance or isolate a specific HTTP error. One folder is generated for each failed request, which contains the XML log file, and the XSL stylesheet to view the log file with.
Deployment logging	Windows, Linux	App Service file system	Logs for when you publish content to an app. Deployment logging happens automatically and there are no configurable settings for deployment logging. It helps you determine why a deployment failed. For example, if you use a custom deployment script , you might use deployment logging to determine why the script is failing.

NOTE

App Service provides a dedicated, interactive diagnostics tool to help you troubleshoot your application. For more information, see [Azure App Service diagnostics overview](#).

In addition, you can use other Azure services to improve the logging and monitoring capabilities of your app, such as [Azure Monitor](#).

Enable application logging (Windows)

NOTE

Application logging for blob storage can only use storage accounts in the same region as the App Service

To enable application logging for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

Select **On** for either **Application Logging (Filesystem)** or **Application Logging (Blob)**, or both.

The **Filesystem** option is for temporary debugging purposes, and turns itself off in 12 hours. The **Blob** option is for long-term logging, and needs a blob storage container to write logs to. The **Blob** option also includes additional information in the log messages, such as the ID of the origin VM instance of the log message (`InstanceId`), thread ID (`Tid`), and a more granular timestamp (`EventTickCount`).

NOTE

Currently only .NET application logs can be written to the blob storage. Java, PHP, Node.js, Python application logs can only be stored on the App Service file system (without code modifications to write logs to external storage).

Also, if you [regenerate your storage account's access keys](#), you must reset the respective logging configuration to use the updated access keys. To do this:

1. In the **Configure** tab, set the respective logging feature to **Off**. Save your setting.
2. Enable logging to the storage account blob again. Save your setting.

Select the **Level**, or the level of details to log. The following table shows the log categories included in each level:

LEVEL	INCLUDED CATEGORIES
Disabled	None
Error	Error, Critical
Warning	Warning, Error, Critical
Information	Info, Warning, Error, Critical
Verbose	Trace, Debug, Info, Warning, Error, Critical (all categories)

When finished, select **Save**.

Enable application logging (Linux/Container)

To enable application logging for Linux apps or custom container apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

In **Application logging**, select **File System**.

In **Quota (MB)**, specify the disk quota for the application logs. In **Retention Period (Days)**, set the number of days the logs should be retained.

When finished, select **Save**.

Enable web server logging

To enable web server logging for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

For **Web server logging**, select **Storage** to store logs on blob storage, or **File System** to store logs on the App Service file system.

In **Retention Period (Days)**, set the number of days the logs should be retained.

NOTE

If you [regenerate your storage account's access keys](#), you must reset the respective logging configuration to use the updated keys. To do this:

1. In the **Configure** tab, set the respective logging feature to **Off**. Save your setting.
2. Enable logging to the storage account blob again. Save your setting.

When finished, select **Save**.

Log detailed errors

To save the error page or failed request tracing for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

Under **Detailed Error Logging** or **Failed Request Tracing**, select **On**, then select **Save**.

Both types of logs are stored in the App Service file system. Up to 50 errors (files/folders) are retained. When the number of HTML files exceed 50, the oldest 26 errors are automatically deleted.

Add log messages in code

In your application code, you use the usual logging facilities to send log messages to the application logs. For example:

- ASP.NET applications can use the [System.Diagnostics.Trace](#) class to log information to the application diagnostics log. For example:

```
System.Diagnostics.Trace.TraceError("If you're seeing this, something bad happened");
```

- By default, ASP.NET Core uses the [Microsoft.Extensions.Logging.AzureAppServices](#) logging provider. For more information, see [ASP.NET Core logging in Azure](#). For information about WebJobs SDK logging, see [Get started with the Azure WebJobs SDK](#)

Stream logs

Before you stream logs in real time, enable the log type that you want. Any information written to files ending in .txt, .log, or .htm that are stored in the `/LogFiles` directory (d:/home/logfiles) is streamed by App Service.

NOTE

Some types of logging buffer write to the log file, which can result in out of order events in the stream. For example, an application log entry that occurs when a user visits a page may be displayed in the stream before the corresponding HTTP log entry for the page request.

In Azure portal

To stream logs in the [Azure portal](#), navigate to your app and select **Log stream**.

In Cloud Shell

To stream logs live in [Cloud Shell](#), use the following command:

IMPORTANT

This command may not work with web apps hosted in a Linux app service plan.

```
az webapp log tail --name appname --resource-group myResourceGroup
```

To filter specific log types, such as HTTP, use the `--Provider` parameter. For example:

```
az webapp log tail --name appname --resource-group myResourceGroup --provider http
```

In local terminal

To stream logs in the local console, [install Azure CLI](#) and [sign in to your account](#). Once signed in, followed the [instructions for Cloud Shell](#)

Access log files

If you configure the Azure Storage blobs option for a log type, you need a client tool that works with Azure Storage. For more information, see [Azure Storage Client Tools](#).

For logs stored in the App Service file system, the easiest way is to download the ZIP file in the browser at:

- Linux/container apps: `https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip`
- Windows apps: `https://<app-name>.scm.azurewebsites.net/api/dump`

For Linux/container apps, the ZIP file contains console output logs for both the docker host and the docker container. For a scaled-out app, the ZIP file contains one set of logs for each instance. In the App Service file system, these log files are the contents of the `/home/LogFiles` directory.

For Windows apps, the ZIP file contains the contents of the `D:\Home\LogFiles` directory in the App Service file system. It has the following structure:

LOG TYPE	DIRECTORY	DESCRIPTION
Application logs	<code>/LogFiles/Application/</code>	Contains one or more text files. The format of the log messages depends on the logging provider you use.
Failed Request Traces	<code>/LogFiles/W3SVC#####/</code>	Contains XML files, and an XSL file. You can view the formatted XML files in the browser.
Detailed Error Logs	<code>/LogFiles/DetailedErrors/</code>	Contains HTM error files. You can view the HTM files in the browser. Another way to view the failed request traces is to navigate to your app page in the portal. From the left menu, select Diagnose and solve problems , then search for Failed Request Tracing Logs , then click the icon to browse and view the trace you want.

LOG TYPE	DIRECTORY	DESCRIPTION
Web Server Logs	<i>/LogFiles/http/RawLogs/</i>	Contains text files formatted using the W3C extended log file format . This information can be read using a text editor or a utility like Log Parser . App Service doesn't support the <code>s-computername</code> , <code>s-ip</code> , or <code>cs-version</code> fields.
Deployment logs	<i>/LogFiles/Git/ and /deployments/</i>	Contain logs generated by the internal deployment processes, as well as logs for Git deployments.

Send logs to Azure Monitor (preview)

With the new [Azure Monitor integration](#), you can [create Diagnostic Settings \(preview\)](#) to send logs to Storage Accounts, Event Hubs and Log Analytics.

The screenshot shows the Azure portal interface. In the left-hand navigation pane, the 'Diagnostic settings' option is highlighted with a red box. The main content area displays the 'Diagnostics settings' configuration page for a resource. At the top, there are dropdown menus for 'Subscription' (set to 'Demo Two Subscription') and 'Resource group' (set to 'appsvc-azmon-rg'). Below these, a breadcrumb trail shows the path: 'Demo Two Subscription > appsvc-azmon-rg > appsvc-azmon-app'. The main section is titled 'Diagnostics settings' and contains a table with columns for 'Name', 'Storage account', and 'Event hub'. The table is currently empty, with the text 'No diagnostic settings defined' below it. A red box highlights the '+ Add diagnostic setting' button. Below this button, a message states: 'Click 'Add Diagnostic setting' above to configure the collection of the following data:'. A bulleted list follows, listing the supported log types: AppServiceHTTPLogs, AppServiceConsoleLogs, AppServiceAppLogs, AppServiceFileAuditLogs, AppServiceAuditLogs, and AllMetrics.

Supported log types

The following table shows the supported log types and descriptions:

LOG TYPE	WINDOWS	WINDOWS CONTAINER	LINUX	LINUX CONTAINER	DESCRIPTION
AppServiceConsoleLogs	Java SE & Tomcat	Yes	Yes	Yes	Standard output and standard error
AppServiceHTTPLogs	Yes	Yes	Yes	Yes	Web server logs

LOG TYPE	WINDOWS	WINDOWS CONTAINER	LINUX	LINUX CONTAINER	DESCRIPTION
AppServiceEnvironmentPlatformLogs	Yes	N/A	Yes	Yes	App Service Environment: scaling, configuration changes, and status logs
AppServiceAuditLogs	Yes	Yes	Yes	Yes	Login activity via FTP and Kudu
AppServiceFileAuditLogs	Yes	Yes	TBA	TBA	File changes made to the site content; only available for Premium tier and above
AppServiceAppLogs	ASP .NET & Tomcat ¹	ASP .NET & Tomcat ¹	Java SE & Tomcat Blessed Images ²	Java SE & Tomcat Blessed Images ²	Application logs
AppServiceIPSecAuditLogs	Yes	Yes	Yes	Yes	Requests from IP Rules
AppServicePlatformLogs	TBA	Yes	Yes	Yes	Container operation logs
AppServiceAntivirusScanAudits	Yes	Yes	Yes	Yes	Anti-virus scan logs using Microsoft Defender; only available for Premium tier

¹ For Tomcat apps, add "TOMCAT_USE_STARTUP_BAT" to the app settings and set it to false or 0. Need to be on the *latest* Tomcat version and use *java.util.logging*.

² For Java SE apps, add "\$WEBSITE_AZMON_PREVIEW_ENABLED" to the app settings and set it to true or to 1.

Next steps

- [Query logs with Azure Monitor](#)
- [How to Monitor Azure App Service](#)
- [Troubleshooting Azure App Service in Visual Studio](#)
- [Analyze app Logs in HDInsight](#)

Tutorial: Monitor a Windows virtual machine in Azure

5/28/2021 • 4 minutes to read • [Edit Online](#)

Azure monitoring uses agents to collect boot and performance data from Azure VMs, store this data in Azure storage, and make it accessible through portal, the Azure PowerShell module, and Azure CLI. Advanced monitoring is delivered with Azure Monitor for VMs by collecting performance metrics, discover application components installed on the VM, and includes performance charts and dependency map.

In this tutorial, you learn how to:

- Enable boot diagnostics on a VM
- View boot diagnostics
- View VM host metrics
- Enable Azure Monitor for VMs
- View VM performance metrics
- Create an alert

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.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/powershell>. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

Create virtual machine

To configure Azure monitoring and update management in this tutorial, you need a Windows VM in Azure. First, set an administrator username and password for the VM with [Get-Credential](#):

```
$cred = Get-Credential
```

Now create the VM with [New-AzVm](#). The following example creates a VM named *myVM* in the *EastUS* location. If they do not already exist, the resource group *myResourceGroupMonitorMonitor* and supporting network resources are created:

```
New-AzVm `
  -ResourceGroupName "myResourceGroupMonitor" `
  -Name "myVM" `
  -Location "East US" `
  -Credential $cred
```

It takes a few minutes for the resources and VM to be created.

View boot diagnostics

As Windows virtual machines boot up, the boot diagnostic agent captures screen output that can be used for

troubleshooting purpose. This capability is enabled by default. The captured screenshots are stored in an Azure storage account, which is also created by default.

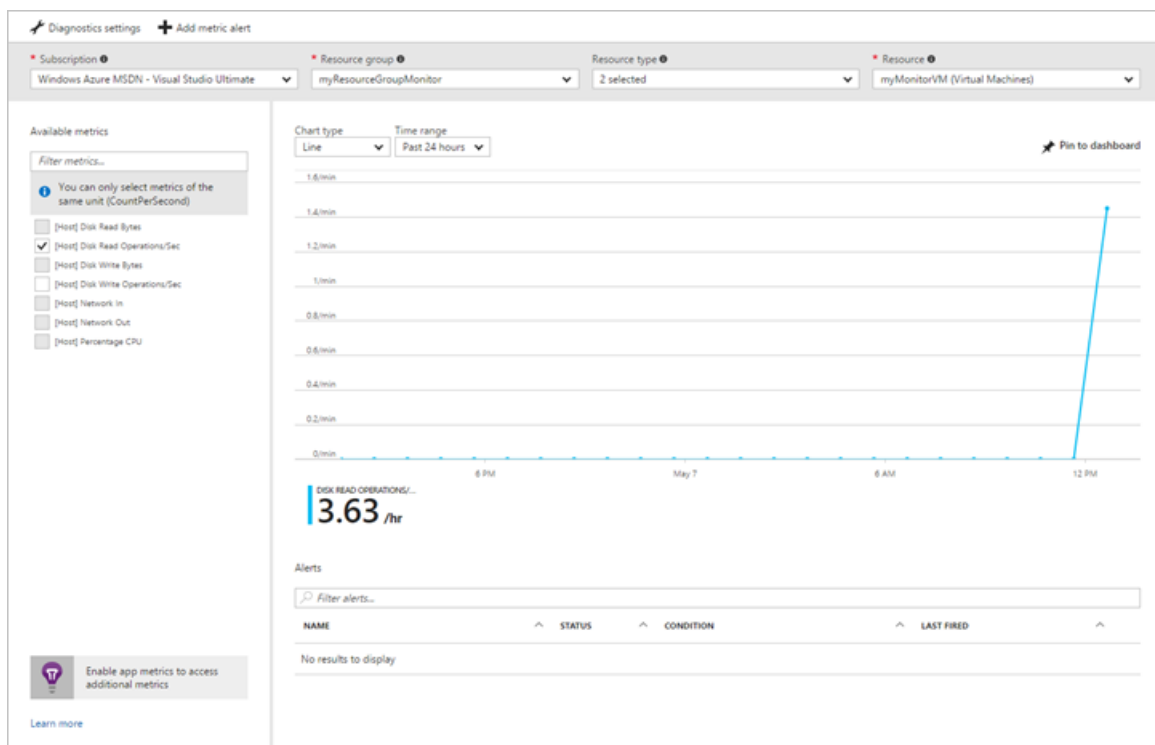
You can get the boot diagnostic data with the [Get-AzureRmVMBootDiagnosticsData](#) command. In the following example, boot diagnostics are downloaded to the root of the *c:* drive.

```
Get-AzVMBootDiagnosticsData -ResourceGroupName "myResourceGroupMonitor" -Name "myVM" -Windows -LocalPath "c:\\"
```

View host metrics

A Windows VM has a dedicated Host VM in Azure that it interacts with. Metrics are automatically collected for the Host and can be viewed in the Azure portal.

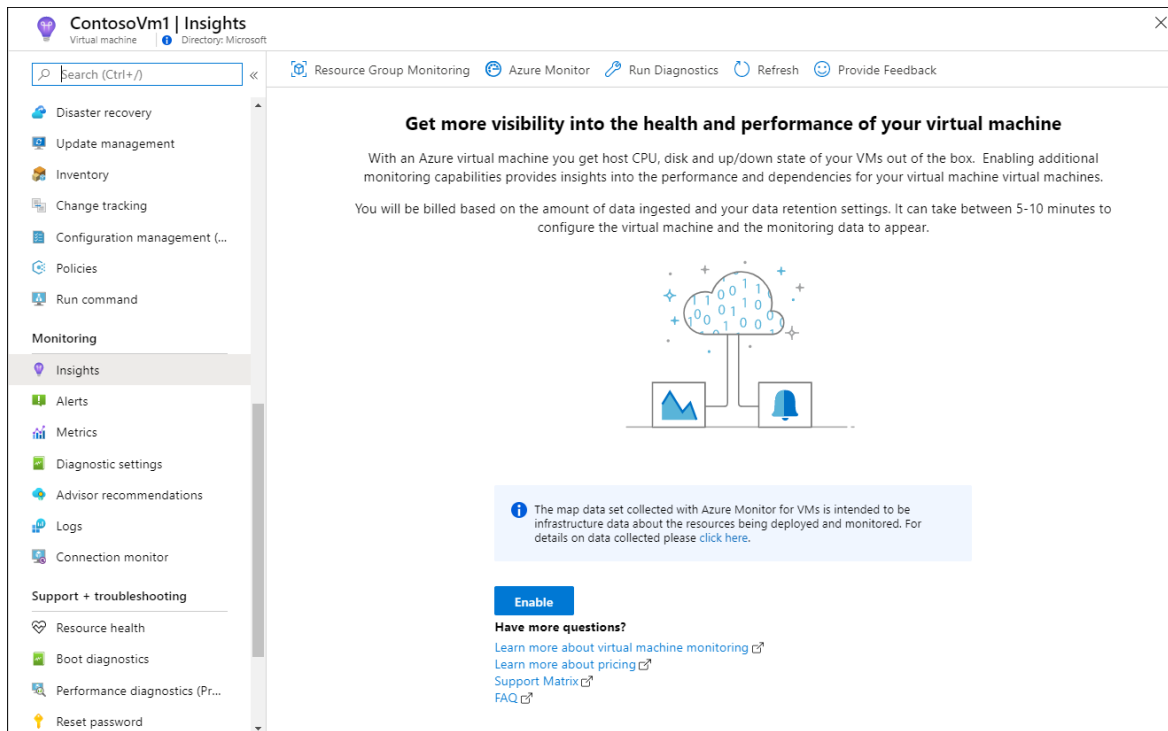
1. In the Azure portal, click **Resource Groups**, select **myResourceGroupMonitor**, and then select **myVM** in the resource list.
2. Click **Metrics** on the VM blade, and then select any of the Host metrics under **Available metrics** to see how the Host VM is performing.



Enable advanced monitoring

To enable monitoring of your Azure VM with Azure Monitor for VMs:

1. In the Azure portal, click **Resource Groups**, select **myResourceGroupMonitor**, and then select **myVM** in the resource list.
2. On the VM page, in the **Monitoring** section, select **Insights (preview)**.
3. On the **Insights (preview)** page, select **Try now**.



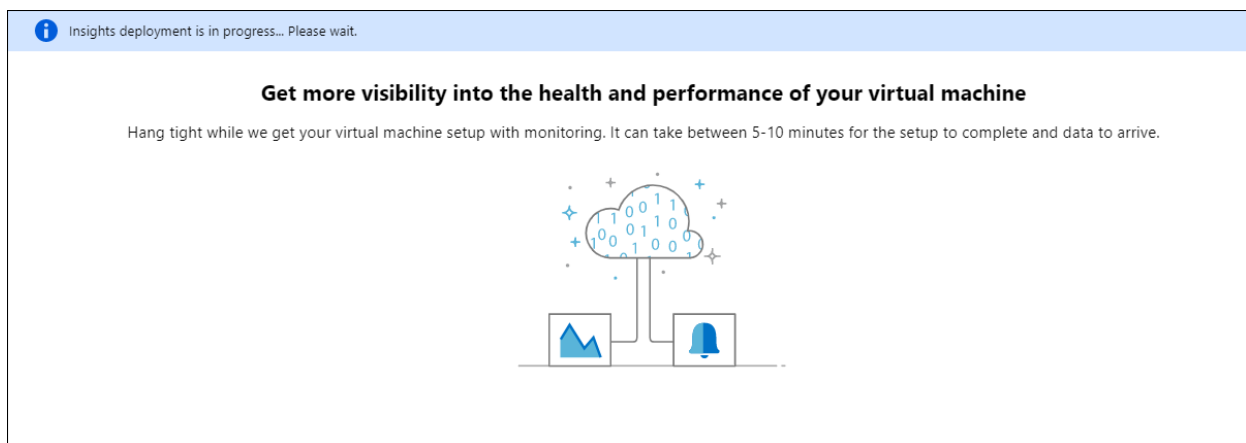
4. On the **Azure Monitor Insights Onboarding** page, if you have an existing Log Analytics workspace in the same subscription, select it in the drop-down list.

The list preselects the default workspace and location where the VM is deployed in the subscription.

NOTE

To create a new Log Analytics workspace to store the monitoring data from the VM, see [Create a Log Analytics workspace](#). The workspace must belong to one of the [supported regions](#).

After you've enabled monitoring, you might need to wait several minutes before you can view the performance metrics for the VM.



View VM performance metrics

Azure Monitor for VMs includes a set of performance charts that target several key performance indicators (KPIs) to help you determine how well a virtual machine is performing. To access from your VM, perform the following steps.

1. In the Azure portal, click **Resource Groups**, select **myResourceGroupMonitor**, and then select **myVM** in the resource list.
2. On the VM page, in the **Monitoring** section, select **Insights (preview)**.

3. Select the **Performance** tab.

This page not only includes performance utilization charts, but also a table showing for each logical disk discovered, its capacity, utilization, and total average by each measure.

Create alerts

You can create alerts based on specific performance metrics. Alerts can be used to notify you when average CPU usage exceeds a certain threshold or available free disk space drops below a certain amount, for example. Alerts are displayed in the Azure portal or can be sent via email. You can also trigger Azure Automation runbooks or Azure Logic Apps in response to alerts being generated.

The following example creates an alert for average CPU usage.

1. In the Azure portal, click **Resource Groups**, select **myResourceGroupMonitor**, and then select **myVM** in the resource list.
2. Click **Alert rules** on the VM blade, then click **Add metric alert** across the top of the alerts blade.
3. Provide a **Name** for your alert, such as *myAlertRule*
4. To trigger an alert when CPU percentage exceeds 1.0 for five minutes, leave all the other defaults selected.
5. Optionally, check the box for *Email owners, contributors, and readers* to send email notification. The default action is to present a notification in the portal.
6. Click the **OK** button.

Next steps

In this tutorial, you configured and viewed performance of your VM. You learned how to:

- Create a resource group and VM
- Enable boot diagnostics on the VM
- View boot diagnostics
- View host metrics
- Enable Azure Monitor for VMs
- View VM metrics
- Create an alert

Advance to the next tutorial to learn about Azure Security Center.

[Manage VM security](#)

Monitoring and diagnostics for Azure Service Fabric

6/22/2021 • 9 minutes to read • [Edit Online](#)

This article provides an overview of monitoring and diagnostics for Azure Service Fabric. Monitoring and diagnostics are critical to developing, testing, and deploying workloads in any cloud environment. For example, you can track how your applications are used, the actions taken by the Service Fabric platform, your resource utilization with performance counters, and the overall health of your cluster. You can use this information to diagnose and correct issues, and prevent them from occurring in the future. The next few sections will briefly explain each area of Service Fabric monitoring to consider for production workloads.

NOTE

This article was recently updated to use the term Azure Monitor logs instead of Log Analytics. Log data is still stored in a Log Analytics workspace and is still collected and analyzed by the same Log Analytics service. We are updating the terminology to better reflect the role of [logs in Azure Monitor](#). See [Azure Monitor terminology changes](#) for details.

Application monitoring

Application monitoring tracks how features and components of your application are being used. You want to monitor your applications to make sure issues that impact users are caught. The responsibility of application monitoring is on the users developing an application and its services since it is unique to the business logic of your application. Monitoring your applications can be useful in the following scenarios:

- How much traffic is my application experiencing? - Do you need to scale your services to meet user demands or address a potential bottleneck in your application?
- Are my service to service calls successful and tracked?
- What actions are taken by the users of my application? - Collecting telemetry can guide future feature development and better diagnostics for application errors
- Is my application throwing unhandled exceptions?
- What is happening within the services running inside my containers?

The great thing about application monitoring is that developers can use whatever tools and framework they'd like since it lives within the context of your application! You can learn more about the Azure solution for application monitoring with Azure Monitor - Application Insights in [Event analysis with Application Insights](#). We also have a tutorial with how to [set this up for .NET Applications](#). This tutorial goes over how to install the right tools, an example to write custom telemetry in your application, and viewing the application diagnostics and telemetry in the Azure portal.

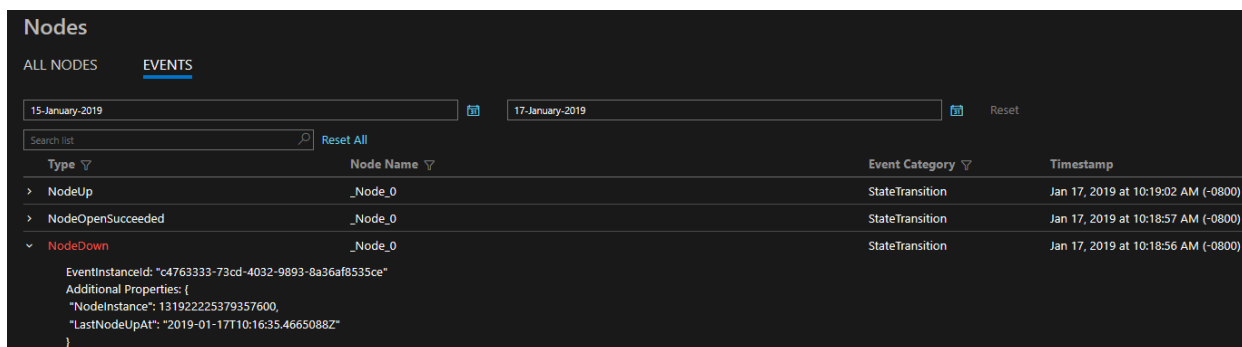
Platform (Cluster) monitoring

A user is in control over what telemetry comes from their application since a user writes the code itself, but what about the diagnostics from the Service Fabric platform? One of Service Fabric's goals is to keep applications resilient to hardware failures. This goal is achieved through the platform's system services' ability to detect infrastructure issues and rapidly failover workloads to other nodes in the cluster. But in this particular case, what if the system services themselves have issues? Or if in attempting to deploy or move a workload, rules for the placement of services are violated? Service Fabric provides diagnostics for these and more to make sure you are informed about activity taking place in your cluster. Some sample scenarios for cluster monitoring include:

Service Fabric provides a comprehensive set of events out of the box. These [Service Fabric events](#) can be

accessed through the EventStore or the operational channel (event channel exposed by the platform).

- Service Fabric event channels - On Windows, Service Fabric events are available from a single ETW provider with a set of relevant `LogLevelKeywordFilters` used to pick between Operational and Data & Messaging channels - this is the way in which we separate out outgoing Service Fabric events to be filtered on as needed. On Linux, Service Fabric events come through LTTng and are put into one Storage table, from where they can be filtered as needed. These channels contain curated, structured events that can be used to better understand the state of your cluster. Diagnostics are enabled by default at the cluster creation time, which create an Azure Storage table where the events from these channels are sent for you to query in the future.
- EventStore - The EventStore is a feature offered by the platform that provides Service Fabric platform events available in the Service Fabric Explorer and through REST API. You can see a snapshot view of what's going on in your cluster for each entity e.g. node, service, application and query based on the time of the event. You can also Read more about the EventStore at the [EventStore Overview](#).



Type	Node Name	Event Category	Timestamp
> NodeUp	_Node_0	StateTransition	Jan 17, 2019 at 10:19:02 AM (-0800)
> NodeOpenSucceeded	_Node_0	StateTransition	Jan 17, 2019 at 10:18:57 AM (-0800)
< NodeDown	_Node_0	StateTransition	Jan 17, 2019 at 10:18:56 AM (-0800)

```
EventInstanceId: "c4763333-73cd-4032-9893-8a36af8535ce"
Additional Properties: {
  "NodeInstance": "131922225379357600",
  "LastNodeUpAt": "2019-01-17T10:16:35.4665088Z"
}
```

The diagnostics provided are in the form of a comprehensive set of events out of the box. These [Service Fabric events](#) illustrate actions done by the platform on different entities such as Nodes, Applications, Services, Partitions etc. In the last scenario above, if a node were to go down, the platform would emit a `NodeDown` event and you could be notified immediately by your monitoring tool of choice. Other common examples include `ApplicationUpgradeRollbackStarted` or `PartitionReconfigured` during a failover. **The same events are available on both Windows and Linux clusters.**

The events are sent through standard channels on both Windows and Linux and can be read by any monitoring tool that supports these. The Azure Monitor solution is Azure Monitor logs. Feel free to read more about our [Azure Monitor logs integration](#) which includes a custom operational dashboard for your cluster and some sample queries from which you can create alerts. More cluster monitoring concepts are available at [Platform level event and log generation](#).

Health monitoring

The Service Fabric platform includes a health model, which provides extensible health reporting for the status of entities in a cluster. Each node, application, service, partition, replica, or instance, has a continuously updatable health status. The health status can either be "OK", "Warning", or "Error". Think of Service Fabric events as verbs done by the cluster to various entities and health as an adjective for each entity. Each time the health of a particular entity transitions, an event will also be emitted. This way you can set up queries and alerts for health events in your monitoring tool of choice, just like any other event.

Additionally, we even let users override health for entities. If your application is going through an upgrade and you have validation tests failing, you can write to Service Fabric Health using the Health API to indicate your application is no longer healthy, and Service Fabric will automatically rollback the upgrade! For more on the health model, check out the [introduction to Service Fabric health monitoring](#)



Watchdogs

Generally, a watchdog is a separate service that watches health and load across services, pings endpoints, and reports unexpected health events in the cluster. This can help prevent errors that may not be detected based only on the performance of a single service. Watchdogs are also a good place to host code that performs remedial actions that don't require user interaction, such as cleaning up log files in storage at certain time intervals. If you want a fully implemented, open source SF watchdog service that includes an easy-to-use watchdog extensibility model and that runs in both Windows and Linux clusters, see the [FabricObserver](#) project. FabricObserver is production-ready software. We encourage you to deploy FabricObserver to your test and production clusters and extend it to meet your needs either through its plug-in model or by forking it and writing your own built-in observers. The former (plug-ins) is the recommended approach.

Infrastructure (performance) monitoring

Now that we've covered the diagnostics in your application and the platform, how do we know the hardware is functioning as expected? Monitoring your underlying infrastructure is a key part of understanding the state of your cluster and your resource utilization. Measuring system performance depends on many factors that can be subjective depending on your workloads. These factors are typically measured through performance counters. These performance counters can come from a variety of sources including the operating system, the .NET framework, or the Service Fabric platform itself. Some scenarios in which they would be useful are

- Am I utilizing my hardware efficiently? Do you want to use your hardware at 90% CPU or 10% CPU. This comes in handy when scaling your cluster, or optimizing your application's processes.
- Can I predict infrastructure issues proactively? - many issues are preceded by sudden changes (drops) in performance, so you can use performance counters such as network I/O and CPU utilization to predict and diagnose the issues proactively.

A list of performance counters that should be collected at the infrastructure level can be found at [Performance metrics](#).

Service Fabric also provides a set of performance counters for the Reliable Services and Actors programming models. If you are using either of these models, these performance counters can information to ensure that your actors are spinning up and down correctly, or that your reliable service requests are being handled fast enough. For more information, see [Monitoring for Reliable Service Remoting](#) and [Performance monitoring for Reliable Actors](#).

The Azure Monitor solution to collect these is Azure Monitor logs just like platform level monitoring. You should use the [Log Analytics agent](#) to collect the appropriate performance counters, and view them in Azure Monitor logs.

Recommended Setup

Now that we've gone over each area of monitoring and example scenarios, here is a summary of the Azure

monitoring tools and set up needed to monitor all areas above.

- Application monitoring with [Application Insights](#)
- Cluster monitoring with [Diagnostics Agent](#) and [Azure Monitor logs](#)
- Infrastructure monitoring with [Azure Monitor logs](#)

You can also use and modify the sample ARM template located [here](#) to automate deployment of all necessary resources and agents.

Other logging solutions

Although the two solutions we recommended, [Azure Monitor logs](#) and [Application Insights](#) have built in integration with Service Fabric, many events are written out through ETW providers and are extensible with other logging solutions. You should also look into the [Elastic Stack](#) (especially if you are considering running a cluster in an offline environment), [Dynatrace](#), or any other platform of your preference. We have a list of integrated partners available [here](#).

The key points for any platform you choose should include how comfortable you are with the user interface, the querying capabilities, the custom visualizations and dashboards available, and the additional tools they provide to enhance your monitoring experience.

Next steps

- For getting started with instrumenting your applications, see [Application level event and log generation](#).
- Go through the steps to set up Application Insights for your application with [Monitor and diagnose an ASP.NET Core application on Service Fabric](#).
- Learn more about monitoring the platform and the events Service Fabric provides for you at [Platform level event and log generation](#).
- Configure the Azure Monitor logs integration with Service Fabric at [Set up Azure Monitor logs for a cluster](#)
- Learn how to set up Azure Monitor logs for monitoring containers - [Monitoring and Diagnostics for Windows Containers in Azure Service Fabric](#).
- See example diagnostics problems and solutions with Service Fabric in [diagnosing common scenarios](#)
- Check out other diagnostics products that integrate with Service Fabric in [Service Fabric diagnostic partners](#)
- Learn about general monitoring recommendations for Azure resources - [Best Practices - Monitoring and diagnostics](#).

Use cost alerts to monitor usage and spending

3/5/2021 • 3 minutes to read • [Edit Online](#)

This article helps you understand and use Cost Management alerts to monitor your Azure usage and spending. Cost alerts are automatically generated based when Azure resources are consumed. Alerts show all active cost management and billing alerts together in one place. When your consumption reaches a given threshold, alerts are generated by Cost Management. There are three types of cost alerts: budget alerts, credit alerts, and department spending quota alerts.

Budget alerts

Budget alerts notify you when spending, based on usage or cost, reaches or exceeds the amount defined in the [alert condition of the budget](#). Cost Management budgets are created using the Azure portal or the [Azure Consumption API](#).

In the Azure portal, budgets are defined by cost. Using the Azure Consumption API, budgets are defined by cost or by consumption usage. Budget alerts support both cost-based and usage-based budgets. Budget alerts are generated automatically whenever the budget alert conditions are met. You can view all cost alerts in the Azure portal. Whenever an alert is generated, it's shown in cost alerts. An alert email is also sent to the people in the alert recipients list of the budget.

You can use the Budget API to send email alerts in a different language. For more information, see [Supported locales for budget alert emails](#).

Credit alerts

Credit alerts notify you when your Azure Prepayment (previously called monetary commitment) is consumed. Azure Prepayment is for organizations with Enterprise Agreements. Credit alerts are generated automatically at 90% and at 100% of your Azure Prepayment credit balance. Whenever an alert is generated, it's reflected in cost alerts and in the email sent to the account owners.

Department spending quota alerts

Department spending quota alerts notify you when department spending reaches a fixed threshold of the quota. Spending quotas are configured in the EA portal. Whenever a threshold is met it generates an email to department owners and is shown in cost alerts. For example, 50% or 75% of the quota.

Supported alert features by offer categories

Support for alert types depends on the type of Azure account that you have (Microsoft offer). The following table shows the alert features that are supported by various Microsoft offers. You can view the full list of Microsoft offers at [Understand Cost Management data](#).

ALERT TYPE	ENTERPRISE AGREEMENT	MICROSOFT CUSTOMER AGREEMENT	WEB DIRECT/PAY-AS-YOU-GO
Budget	✓	✓	✓
Credit	✓	X	X

ALERT TYPE	ENTERPRISE AGREEMENT	MICROSOFT CUSTOMER AGREEMENT	WEB DIRECT/PAY-AS-YOU-GO
Department spending quota	✓	X	X

View cost alerts

To view cost alerts, open the desired scope in the Azure portal and select **Budgets** in the menu. Use the **Scope** pill to switch to a different scope. Select **Cost alerts** in the menu. For more information about scopes, see [Understand and work with scopes](#).

The screenshot shows the Azure portal interface for 'Cost Management: Trey Research - Cost alerts'. The left sidebar contains navigation options like 'Home', 'Dashboard', 'All services', and 'Cost alerts'. The main content area displays a list of active alerts with columns for Type, Name, Date, Status, and Scope. An 'Alert Details' modal is open, showing a description of a budget alert: 'The cost is greater than \$5,400 and exceeds PresupuestolInnovacion threshold of 90%'. It also shows the current cost at 92% (\$5,538.94) and the last update time as February 7, 2019, 8:00 AM. The details section includes fields for Name (Actual greater than 90%), Date (2/7/2019, 8:00:35 AM), Scope (Trey Research (ManagementGroup) > Trey Research Finance (Subscription)), Budget (PresupuestolInnovacion), Period (January 2019), and Email sent to (user@contoso.com).

The total number of active and dismissed alerts appears on the cost alerts page.

All alerts show the alert type. A budget alert shows the reason why it was generated and the name of the budget it applies to. Each alert shows the date it was generated, its status, and the scope (subscription or management group) that the alert applies to.

Possible status includes **active** and **dismissed**. Active status indicates that the alert is still relevant. Dismissed status indicates that someone has marked the alert to set it as no longer relevant.

Select an alert from the list to view its details. Alert details show more information about the alert. Budget alerts include a link to the budget. If a recommendation is available for a budget alert, then a link to the recommendation is also shown. Budget, credit, and department spending quota alerts have a link to analyze in cost analysis where you can explore costs for the alert's scope. The following example shows spending for a department with alert details.

Refresh | Dismiss | Re-Activate

Scope : Contoso (Demo) (8608480) | Filter by name... | All time | Group by : None | Status : 2 selected

Active alerts: 3 | Dismissed alerts: 1

TYPE	NAME	DATE	STATUS	SCOPE
<input type="checkbox"/>	Department spending quota	Spend reached 100% (ACE)	Active	ACE (Department)
<input type="checkbox"/>	Department spending quota	Spend reached 100% (ACM)	Dismissed	ACM (Department)
<input type="checkbox"/>	Department spending quota	Spend reached 100% (DCX Program)	Active	DCX Program (Department)
<input type="checkbox"/>	Azure credit (system notification)	You have used over 100% of your azure credits	Active	Contoso (Demo) (8608480) (Billing account)

Alert Details

Description: You have reached your department ACE spending quota 100% threshold. Spending quota: \$1,000 | [View in EA portal](#)

Current cost: 2193% | \$21,931.59 | Last Update: February 4, 2019, 5:00 PM

Recommendation: [Analyze in cost analysis](#)

Name: Spend reached 100% (ACE) | Date: 1/15/2019, 5:02:51 PM | Scope: Contoso (Demo) (8608480) (BillingAccount) > ACE (Department)

When you view the details of a dismissed alert, you can reactivate it if manual action is needed. The following image shows an example.

Refresh | **Dismiss** | Re-Activate

Scope : Trey Research Finance | Filter by name... | Last 30 days

Active alerts: 4 | Dismissed alerts: 1

TYPE	NAME
<input checked="" type="checkbox"/>	Budget - cost exceeded threshold Actual greater than 100% (PresupuestoInnovacion, January 2019)
<input type="checkbox"/>	Budget - cost exceeded threshold Actual greater than 50% (MonthlyBudget2, January 2019)
<input type="checkbox"/>	Budget - cost exceeded threshold Actual greater than 90% (MonthlyBudget1, January 2019)
<input type="checkbox"/>	Budget - cost exceeded threshold Actual greater than 90% (TeamBudget, January 2019)
<input checked="" type="checkbox"/>	Budget - cost exceeded threshold Actual greater than 90% (Monthly, January 2019)

See also

- If you haven't already created a budget or set alert conditions for a budget, complete the [Create and manage budgets](#) tutorial.

Tutorial: Create a virtual machine scale set and deploy a highly available app on Linux with the Azure CLI

5/4/2021 • 7 minutes to read • [Edit Online](#)

A virtual machine scale set allows you to deploy and manage a set of identical, auto-scaling virtual machines. You can scale the number of VMs in the scale set manually, or define rules to autoscale based on resource usage such as CPU, memory demand, or network traffic. In this tutorial, you deploy a virtual machine scale set in Azure. You learn how to:

- Use cloud-init to create an app to scale
- Create a virtual machine scale set
- Increase or decrease the number of instances in a scale set
- Create autoscale rules
- View connection info for scale set instances
- Use data disks in a scale set

This tutorial uses the CLI within the [Azure Cloud Shell](#), which is constantly updated to the latest version. To open the Cloud Shell, select **Try it** from the top of any code block.

If you choose to install and use the CLI locally, this tutorial requires that you are running the Azure CLI version 2.0.30 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

Scale Set overview

A virtual machine scale set allows you to deploy and manage a set of identical, auto-scaling virtual machines. VMs in a scale set are distributed across logic fault and update domains in one or more *placement groups*. These are groups of similarly configured VMs, similar to [availability sets](#).

VMs are created as needed in a scale set. You define autoscale rules to control how and when VMs are added or removed from the scale set. These rules can be triggered based on metrics such as CPU load, memory usage, or network traffic.

Scale sets support up to 1,000 VMs when you use an Azure platform image. For workloads with significant installation or VM customization requirements, you may wish to [Create a custom VM image](#). You can create up to 300 VMs in a scale set when using a custom image.

Create an app to scale

For production use, you may wish to [Create a custom VM image](#) that includes your application installed and configured. For this tutorial, let's customize the VMs on first boot to quickly see a scale set in action.

In a previous tutorial, you learned [How to customize a Linux virtual machine on first boot](#) with cloud-init. You can use the same cloud-init configuration file to install NGINX and run a simple 'Hello World' Node.js app.

In your current shell, create a file named `cloud-init.txt` and paste the following configuration. For example, create the file in the Cloud Shell not on your local machine. Enter `sensible-editor cloud-init.txt` to create the file and see a list of available editors. Make sure that the whole cloud-init file is copied correctly, especially the first line:

```

#cloud-config
package_upgrade: true
packages:
  - nginx
  - nodejs
  - npm
write_files:
  - owner: www-data:www-data
  - path: /etc/nginx/sites-available/default
    content: |
      server {
        listen 80;
        location / {
          proxy_pass http://localhost:3000;
          proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection keep-alive;
          proxy_set_header Host $host;
          proxy_cache_bypass $http_upgrade;
        }
      }
  - owner: azureuser:azureuser
  - path: /home/azureuser/myapp/index.js
    content: |
      var express = require('express')
      var app = express()
      var os = require('os');
      app.get('/', function (req, res) {
        res.send('Hello World from host ' + os.hostname() + '!')
      })
      app.listen(3000, function () {
        console.log('Hello world app listening on port 3000!')
      })
runcmd:
  - service nginx restart
  - cd "/home/azureuser/myapp"
  - npm init
  - npm install express -y
  - nodejs index.js

```

Create a scale set

Before you can create a scale set, create a resource group with [az group create](#). The following example creates a resource group named *myResourceGroupScaleSet* in the *eastus* location:

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

Now create a virtual machine scale set with [az vmss create](#). The following example creates a scale set named *myScaleSet*, uses the cloud-init file to customize the VM, and generates SSH keys if they do not exist:

```

az vmss create \
  --resource-group myResourceGroupScaleSet \
  --name myScaleSet \
  --image UbuntuLTS \
  --upgrade-policy-mode automatic \
  --custom-data cloud-init.txt \
  --admin-username azureuser \
  --generate-ssh-keys

```

It takes a few minutes to create and configure all the scale set resources and VMs. There are background tasks that continue to run after the Azure CLI returns you to the prompt. It may be another couple of minutes before

you can access the app.

Allow web traffic

A load balancer was created automatically as part of the virtual machine scale set. The load balancer distributes traffic across a set of defined VMs using load balancer rules. You can learn more about load balancer concepts and configuration in the next tutorial, [How to load balance virtual machines in Azure](#).

To allow traffic to reach the web app, create a rule with `az network lb rule create`. The following example creates a rule named `myLoadBalancerRuleWeb`.

```
az network lb rule create \  
  --resource-group myResourceGroupScaleSet \  
  --name myLoadBalancerRuleWeb \  
  --lb-name myScaleSetLB \  
  --backend-pool-name myScaleSetLBBackendPool \  
  --backend-port 80 \  
  --frontend-ip-name loadBalancerFrontEnd \  
  --frontend-port 80 \  
  --protocol tcp
```

Test your app

To see your Node.js app on the web, obtain the public IP address of your load balancer with `az network public-ip show`. The following example obtains the IP address for `myScaleSetLBPublicIP` created as part of the scale set:

```
az network public-ip show \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSetLBPublicIP \  
  --query [ipAddress] \  
  --output tsv
```

Enter the public IP address in to a web browser. The app is displayed, including the hostname of the VM that the load balancer distributed traffic to:



To see the scale set in action, you can force-refresh your web browser to see the load balancer distribute traffic across all the VMs running your app.

Management tasks

Throughout the lifecycle of the scale set, you may need to run one or more management tasks. Additionally, you may want to create scripts that automate various lifecycle-tasks. The Azure CLI provides a quick way to do those tasks. Here are a few common tasks.

View VMs in a scale set

To view a list of VMs running in your scale set, use `az vmss list-instances` as follows:


```
az vmss list-instances \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSet \  
  --output table
```

The output is similar to the following example:

InstanceId	LatestModelApplied	Location	Name	ProvisioningState	ResourceGroup
----- VmId -----					
1	True	eastus	myScaleSet_1	Succeeded	MYRESOURCEGROUPSCALESET
c72ddc34-6c41-4a53-b89e-dd24f27b30ab					
3	True	eastus	myScaleSet_3	Succeeded	MYRESOURCEGROUPSCALESET
44266022-65c3-49c5-92dd-88ffa64f95da					

Manually increase or decrease VM instances

To see the number of instances you currently have in a scale set, use [az vmss show](#) and query on *sku.capacity*:

```
az vmss show \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSet \  
  --query [sku.capacity] \  
  --output table
```

You can then manually increase or decrease the number of virtual machines in the scale set with [az vmss scale](#). The following example sets the number of VMs in your scale set to 3:

```
az vmss scale \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSet \  
  --new-capacity 3
```

Get connection info

To obtain connection information about the VMs in your scale sets, use [az vmss list-instance-connection-info](#). This command outputs the public IP address and port for each VM that allows you to connect with SSH:

```
az vmss list-instance-connection-info \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSet
```

Use data disks with scale sets

You can create and use data disks with scale sets. In a previous tutorial, you learned how to [Manage Azure disks](#) that outlines the best practices and performance improvements for building apps on data disks rather than the OS disk.

Create scale set with data disks

To create a scale set and attach data disks, add the `--data-disk-sizes-gb` parameter to the [az vmss create](#) command. The following example creates a scale set with 50Gb data disks attached to each instance:

```
az vmss create \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSetDisks \  
  --image UbuntuLTS \  
  --upgrade-policy-mode automatic \  
  --custom-data cloud-init.txt \  
  --admin-username azureuser \  
  --generate-ssh-keys \  
  --data-disk-sizes-gb 50
```

When instances are removed from a scale set, any attached data disks are also removed.

Add data disks

To add a data disk to instances in your scale set, use [az vmss disk attach](#). The following example adds a 50Gb disk to each instance:

```
az vmss disk attach \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSet \  
  --size-gb 50 \  
  --lun 2
```

Detach data disks

To remove a data disk to instances in your scale set, use [az vmss disk detach](#). The following example removes the data disk at LUN 2 from each instance:

```
az vmss disk detach \  
  --resource-group myResourceGroupScaleSet \  
  --name myScaleSet \  
  --lun 2
```

Next steps

In this tutorial, you created a virtual machine scale set. You learned how to:

- Use cloud-init to create an app to scale
- Create a virtual machine scale set
- Increase or decrease the number of instances in a scale set
- Create autoscale rules
- View connection info for scale set instances
- Use data disks in a scale set

Advance to the next tutorial to learn more about load balancing concepts for virtual machines.

[Load balance virtual machines](#)

Tutorial: Create a virtual machine scale set and deploy a highly available app on Windows with Azure PowerShell

11/2/2020 • 7 minutes to read • [Edit Online](#)

A virtual machine scale set allows you to deploy and manage a set of identical, autoscaling virtual machines. You can scale the number of VMs in the scale set manually. You can also define rules to autoscale based on resource usage such as CPU, memory demand, or network traffic. In this tutorial, you deploy a virtual machine scale set in Azure and learn how to:

- Use the Custom Script Extension to define an IIS site to scale
- Create a load balancer for your scale set
- Create a virtual machine scale set
- Increase or decrease the number of instances in a scale set
- Create autoscale rules

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.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/powershell>. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

Scale Set overview

A virtual machine scale set allows you to deploy and manage a set of identical, autoscaling virtual machines. VMs in a scale set are distributed across logic fault and update domains in one or more *placement groups*. Placement groups are groups of similarly configured VMs, similar to [availability sets](#).

VMs are created as needed in a scale set. You define autoscale rules to control how and when VMs are added or removed from the scale set. These rules can trigger based on metrics such as CPU load, memory usage, or network traffic.

Scale sets support up to 1,000 VMs when you use an Azure platform image. For workloads with significant installation or VM customization requirements, you may wish to [Create a custom VM image](#). You can create up to 600 VMs in a scale set when using a custom image.

Create a scale set

Create a virtual machine scale set with [New-AzVmss](#). The following example creates a scale set named *myScaleSet* that uses the *Windows Server 2016 Datacenter* platform image. The Azure network resources for virtual network, public IP address, and load balancer are automatically created. When prompted, you can set your own administrative credentials for the VM instances in the scale set:

```
New-AzVmss `
-ResourceGroupName "myResourceGroupScaleSet" `
-Location "EastUS" `
-VMSScaleSetName "myScaleSet" `
-VirtualNetworkName "myVnet" `
-SubnetName "mySubnet" `
-PublicIpAddressName "myPublicIpAddress" `
-LoadBalancerName "myLoadBalancer" `
-UpgradePolicyMode "Automatic"
```

It takes a few minutes to create and configure all the scale set resources and VMs.

Deploy sample application

To test your scale set, install a basic web application. The Azure Custom Script Extension is used to download and run a script that installs IIS on the VM instances. This extension is useful for post deployment configuration, software installation, or any other configuration / management task. For more information, see the [Custom Script Extension overview](#).

Use the Custom Script Extension to install a basic IIS web server. Apply the Custom Script Extension that installs IIS as follows:

```
# Define the script for your Custom Script Extension to run
$publicSettings = @{
    "fileUri" = (,"https://raw.githubusercontent.com/Azure-Samples/compute-automation-
configurations/master/automate-iis.ps1");
    "commandToExecute" = "powershell -ExecutionPolicy Unrestricted -File automate-iis.ps1"
}

# Get information about the scale set
$vmss = Get-AzVmss `
-ResourceGroupName "myResourceGroupScaleSet" `
-VMSScaleSetName "myScaleSet"

# Use Custom Script Extension to install IIS and configure basic website
Add-AzVmssExtension -VirtualMachineScaleSet $vmss `
-Name "customScript" `
-Publisher "Microsoft.Compute" `
-Type "CustomScriptExtension" `
-TypeHandlerVersion 1.8 `
-Setting $publicSettings

# Update the scale set and apply the Custom Script Extension to the VM instances
Update-AzVmss `
-ResourceGroupName "myResourceGroupScaleSet" `
-Name "myScaleSet" `
-VirtualMachineScaleSet $vmss
```

Allow traffic to application

To allow access to the basic web application, create a network security group with [New-AzNetworkSecurityRuleConfig](#) and [New-AzNetworkSecurityGroup](#). For more information, see [Networking for Azure virtual machine scale sets](#).

```

# Get information about the scale set
$vmss = Get-AzVmss `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -VMSScaleSetName "myScaleSet"

#Create a rule to allow traffic over port 80
$nsgFrontendRule = New-AzNetworkSecurityRuleConfig `
  -Name myFrontendNSGRule `
  -Protocol Tcp `
  -Direction Inbound `
  -Priority 200 `
  -SourceAddressPrefix * `
  -SourcePortRange * `
  -DestinationAddressPrefix * `
  -DestinationPortRange 80 `
  -Access Allow

#Create a network security group and associate it with the rule
$nsgFrontend = New-AzNetworkSecurityGroup `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -Location EastUS `
  -Name myFrontendNSG `
  -SecurityRules $nsgFrontendRule

$vnnet = Get-AzVirtualNetwork `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -Name myVnet

$frontendSubnet = $vnnet.Subnets[0]

$frontendSubnetConfig = Set-AzVirtualNetworkSubnetConfig `
  -VirtualNetwork $vnnet `
  -Name mySubnet `
  -AddressPrefix $frontendSubnet.AddressPrefix `
  -NetworkSecurityGroup $nsgFrontend

Set-AzVirtualNetwork -VirtualNetwork $vnnet

# Update the scale set and apply the Custom Script Extension to the VM instances
Update-AzVmss `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -Name "myScaleSet" `
  -VirtualMachineScaleSet $vmss

```

Test your scale set

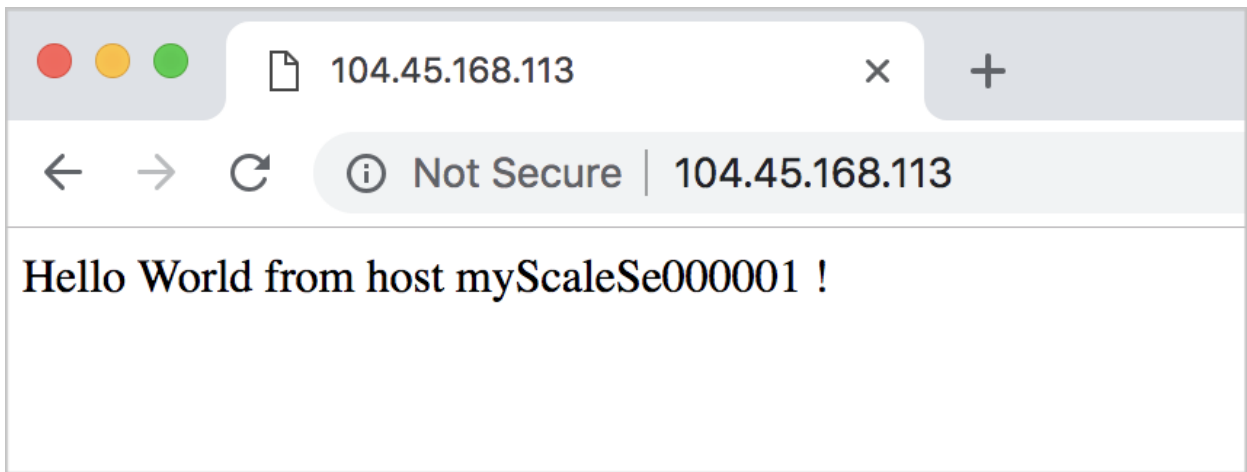
To see your scale set in action, get the public IP address of your load balancer with [Get-AzPublicIPAddress](#). The following example displays the IP address for *myPublicIP* created as part of the scale set:

```

Get-AzPublicIPAddress `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -Name "myPublicIPAddress" | select IPAddress

```

Enter the public IP address in to a web browser. The web app is displayed, including the hostname of the VM that the load balancer distributed traffic to:



To see the scale set in action, you can force-refresh your web browser to see the load balancer distribute traffic across all the VMs running your app.

Management tasks

Throughout the lifecycle of the scale set, you may need to run one or more management tasks. Additionally, you may want to create scripts that automate various lifecycle-tasks. Azure PowerShell provides a quick way to do those tasks. Here are a few common tasks.

View VMs in a scale set

To view a list of VM instances in a scale set, use [Get-AzVmssVM](#) as follows:

```
Get-AzVmssVM `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -VMScaleSetName "myScaleSet"
```

The following example output shows two VM instances in the scale set:

ResourceGroupName	Name	Location	Skus	InstanceID	ProvisioningState
MYRESOURCEGROUPSCALESET	myScaleSet_0	eastus	Standard_DS1_v2	0	Succeeded
MYRESOURCEGROUPSCALESET	myScaleSet_1	eastus	Standard_DS1_v2	1	Succeeded

To view additional information about a specific VM instance, add the `-InstanceId` parameter to [Get-AzVmssVM](#). The following example views information about VM instance `1`:

```
Get-AzVmssVM `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -VMScaleSetName "myScaleSet" `
  -InstanceId "1"
```

Increase or decrease VM instances

To see the number of instances you currently have in a scale set, use [Get-AzVmss](#) and query on `sku.capacity`.

```
Get-AzVmss -ResourceGroupName "myResourceGroupScaleSet" `
  -VMScaleSetName "myScaleSet" | `
  Select -ExpandProperty Sku
```

You can then manually increase or decrease the number of virtual machines in the scale set with [Update-AzVmss](#). The following example sets the number of VMs in your scale set to `3`:

```
# Get current scale set
$scaleset = Get-AzVmss `
  -ResourceGroupName "myResourceGroupScaleSet" `
  -VMSScaleSetName "myScaleSet"

# Set and update the capacity of your scale set
$scaleset.sku.capacity = 3
Update-AzVmss -ResourceGroupName "myResourceGroupScaleSet" `
  -Name "myScaleSet" `
  -VirtualMachineScaleSet $scaleset
```

It takes a few minutes to update the specified number of instances in your scale set.

Configure autoscale rules

Rather than manually scaling the number of instances in your scale set, you define autoscale rules. These rules monitor the instances in your scale set and respond accordingly based on metrics and thresholds you define. The following example scales out the number of instances by one when the average CPU load is greater than 60% over a 5-minute period. If the average CPU load then drops below 30% over a 5-minute period, the instances are scaled in by one instance:

```

# Define your scale set information
$mySubscriptionId = (Get-AzSubscription)[0].Id
$myResourceGroup = "myResourceGroupScaleSet"
$myScaleSet = "myScaleSet"
$myLocation = "East US"
$myScaleSetId = (Get-AzVmss -ResourceGroupName $myResourceGroup -VMSScaleSetName $myScaleSet).Id

# Create a scale up rule to increase the number instances after 60% average CPU usage exceeded for a 5-
minute period
$myRuleScaleUp = New-AzAutoscaleRule `
  -MetricName "Percentage CPU" `
  -MetricResourceId $myScaleSetId `
  -Operator GreaterThan `
  -MetricStatistic Average `
  -Threshold 60 `
  -TimeGrain 00:01:00 `
  -TimeWindow 00:05:00 `
  -ScaleActionCooldown 00:05:00 `
  -ScaleActionDirection Increase `
  -ScaleActionValue 1

# Create a scale down rule to decrease the number of instances after 30% average CPU usage over a 5-minute
period
$myRuleScaleDown = New-AzAutoscaleRule `
  -MetricName "Percentage CPU" `
  -MetricResourceId $myScaleSetId `
  -Operator LessThan `
  -MetricStatistic Average `
  -Threshold 30 `
  -TimeGrain 00:01:00 `
  -TimeWindow 00:05:00 `
  -ScaleActionCooldown 00:05:00 `
  -ScaleActionDirection Decrease `
  -ScaleActionValue 1

# Create a scale profile with your scale up and scale down rules
$myScaleProfile = New-AzAutoscaleProfile `
  -DefaultCapacity 2 `
  -MaximumCapacity 10 `
  -MinimumCapacity 2 `
  -Rule $myRuleScaleUp,$myRuleScaleDown `
  -Name "autoprofile"

# Apply the autoscale rules
Add-AzAutoscaleSetting `
  -Location $myLocation `
  -Name "autosetting" `
  -ResourceGroup $myResourceGroup `
  -TargetResourceId $myScaleSetId `
  -AutoscaleProfile $myScaleProfile

```

For more design information on the use of autoscale, see [autoscale best practices](#).

Next steps

In this tutorial, you created a virtual machine scale set. You learned how to:

- Use the Custom Script Extension to define an IIS site to scale
- Create a load balancer for your scale set
- Create a virtual machine scale set
- Increase or decrease the number of instances in a scale set
- Create autoscale rules

Advance to the next tutorial to learn more about load balancing concepts for virtual machines.

[Load balance virtual machines](#)

Azure consumption API overview

3/5/2021 • 8 minutes to read • [Edit Online](#)

The Azure Consumption APIs give you programmatic access to cost and usage data for your Azure resources. These APIs currently only support Enterprise Enrollments and Web Direct Subscriptions (with a few exceptions). The APIs are continually updated to support other types of Azure subscriptions.

Azure Consumption APIs provide access to:

- Enterprise and Web Direct Customers
 - Usage Details
 - Marketplace Charges
 - Reservation Recommendations
 - Reservation Details
 - Reservation Summaries
- Enterprise Customers Only
 - Price sheet
 - Budgets
 - Balances

Usage Details API

Use the Usage Details API to get charge and usage data for all Azure 1st party resources. Information is in the form of usage detail records which are currently emitted once per meter per resource per day. Information can be used to add up the costs across all resources or investigate costs / usage on specific resource(s).

The API includes:

- **Meter Level Consumption Data** - See data including usage cost, the meter emitting the charge, and what Azure resource the charge pertains to. All usage detail records map to a daily bucket.
- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Filtering** - Trim your API result set down to a smaller set of usage detail records using the following filters: - Usage end / usage start - Resource Group - Resource Name
- **Data Aggregation** - Use OData to apply expressions to aggregate usage details by tags or filter properties
- **Usage for different offer types** - Usage detail information is currently available for Enterprise and Web Direct customers.

For more information, see the technical specification for the [Usage Details API](#).

Marketplace Charges API

Use the Marketplace Charges API to get charge and usage data on all Marketplace resources (Azure 3rd party offerings). This data can be used to add up costs across all Marketplace resources or investigate costs / usage on specific resource(s).

The API includes:

- **Meter Level Consumption Data** - See data including marketplace usage cost, the meter emitting the charge, and what resource the charge pertains to. All usage detail records map to a daily bucket.
- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Filtering** - Trim your API result set down to a smaller set of marketplace records using the following filters: - Usage start / usage end - Resource Group - Resource Name
- **Usage for different offer types** - Marketplace information is currently available for Enterprise and Web Direct customers.

For more information, see the technical specification for the [Marketplace Charges API](#).

Balances API

Enterprise customers can use the Balances API to get a monthly summary of information on balances, new purchases, Azure Marketplace service charges, adjustments, and overage charges. You can get this information for the current billing period or any period in the past. Enterprises can use this data to perform a comparison with manually calculated summary charges. This API does not provide resource-specific information and an aggregate view of costs.

The API includes:

- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Enterprise Customers Only** This API is only available EA customers. - Customers must have Enterprise Admin permissions to call this API

For more information, see the technical specification for the [Balances API](#).

Budgets API

Enterprise customers can use this API to create either cost or usage budgets for resources, resource groups, or billing meters. Once this information has been determined, alerting can be configured to notify when user-defined budget thresholds are exceeded.

The API includes:

- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Enterprise Customers Only** - This API is only available EA customers.
- **Configurable Notifications** - Specify user(s) to be notified when the budget is tripped.
- **Usage or Cost Based Budgets** - Create your budget based on either consumption or cost as needed by your scenario.
- **Filtering** - Filter your budget to a smaller subset of resources using the following configurable filters - Resource Group - Resource Name - Meter
- **Configurable budget time periods** - Specify how often the budget should reset and how long the budget

is valid for.

For more information, see the technical specification for the [Budgets API](#).

Reservation Recommendations API

Use this API to get recommendations for purchasing Reserved VM Instances. Recommendations are designed to allow customers to analyze expected cost savings and purchase amounts.

The API includes:

- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Filtering** - Tailor your recommendation results using the following filters: - Scope - Lookback period
- **Reservation info for different offer types** - Reservation information is currently available for Enterprise and Web Direct customers.

For more information, see the technical specification for the [Reservation Recommendations API](#).

Reservation Details API

Use the Reservation Details API to see info on previously purchased VM reservations such as how much consumption has been reserved versus how much is actually being used. You can see data at a per VM level detail.

The API includes:

- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Filtering** - Trim your API result set down to a smaller set of reservations using the following filter: - Date range
- **Reservation info for different offer types** - Reservation information is currently available for Enterprise and Web Direct customers.

For more information, see the technical specification for the [Reservation Details API](#).

Reservation Summaries API

Use this API to see aggregate information on previously purchased VM reservations such as how much consumption has been reserved versus how much is actually being used in the aggregate.

The API includes:

- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Filtering** - Tailor your results when using the daily grain with the following filter: - Usage Date
- **Reservation info for different offer types** - Reservation information is currently available for Enterprise

and Web Direct customers.

- **Daily or monthly aggregations** – Callers can specify whether they want their reservation summary data in the daily or monthly grain.

For more information, see the technical specification for the [Reservation Summaries API](#).

Price Sheet API

Enterprise customer can use this API to retrieve their custom pricing for all meters. Enterprises can use this in combination with usage details and marketplaces usage info to perform cost calculations using usage and marketplace data.

The API includes:

- **Azure role-based access control (Azure RBAC)** - Configure access policies on the [Azure portal](#), the [Azure CLI](#) or [Azure PowerShell cmdlets](#) to specify which users or applications can get access to the subscription's usage data. Callers must use standard Azure Active Directory tokens for authentication. Add the caller to either the Billing Reader, Reader, Owner, or Contributor role to get access to the usage data for a specific Azure subscription.
- **Enterprise Customers Only** - This API is only available EA customers. Web Direct customers should use the RateCard API to get pricing.

For more information, see the technical specification for the [Price Sheet API](#).

Scenarios

Here are some of the scenarios that are made possible via the consumption APIs:

- **Invoice Reconciliation** - Did Microsoft charge me the right amount? What is my bill and can I calculate it myself?
- **Cross Charges** - Now that I know how much I'm being charged, who in my org needs to pay?
- **Cost Optimization** - I know how much I've been charged... how can I get more out of the money I am spending on Azure?
- **Cost Tracking** - I want to see how much I am spending and using Azure over time. What are the trends? How could I be doing better?
- **Azure spend during the month** - How much is my current month's spend to date? Do I need to make any adjustments in my spending and/or usage of Azure? When during the month am I consuming Azure the most?
- **Set up alerts** - I would like to set up resource-based consumption or monetary-based alerting based on a budget.

Next Steps

- For information about using REST APIs retrieve prices for all Azure services, see [Azure Retail Prices overview](#).

Azure subscription and service limits, quotas, and constraints

6/24/2021 • 112 minutes to read • [Edit Online](#)

This document lists some of the most common Microsoft Azure limits, which are also sometimes called quotas.

To learn more about Azure pricing, see [Azure pricing overview](#). There, you can estimate your costs by using the [pricing calculator](#). You also can go to the pricing details page for a particular service, for example, [Windows VMs](#). For tips to help manage your costs, see [Prevent unexpected costs with Azure billing and cost management](#).

Managing limits

NOTE

Some services have adjustable limits.

When a service doesn't have adjustable limits, the following tables use the header **Limit**. In those cases, the default and the maximum limits are the same.

When the limit can be adjusted, the tables include **Default limit** and **Maximum limit** headers. The limit can be raised above the default limit but not above the maximum limit.

If you want to raise the limit or quota above the default limit, [open an online customer support request at no charge](#).

The terms *soft limit* and *hard limit* often are used informally to describe the current, adjustable limit (soft limit) and the maximum limit (hard limit). If a limit isn't adjustable, there won't be a soft limit, only a hard limit.

[Free Trial subscriptions](#) aren't eligible for limit or quota increases. If you have a [Free Trial subscription](#), you can upgrade to a [Pay-As-You-Go](#) subscription. For more information, see [Upgrade your Azure Free Trial subscription to a Pay-As-You-Go subscription](#) and the [Free Trial subscription FAQ](#).

Some limits are managed at a regional level.

Let's use vCPU quotas as an example. To request a quota increase with support for vCPUs, you must decide how many vCPUs you want to use in which regions. You then request an increase in vCPU quotas for the amounts and regions that you want. If you need to use 30 vCPUs in West Europe to run your application there, you specifically request 30 vCPUs in West Europe. Your vCPU quota isn't increased in any other region--only West Europe has the 30-vCPU quota.

As a result, decide what your quotas must be for your workload in any one region. Then request that amount in each region into which you want to deploy. For help in how to determine your current quotas for specific regions, see [Resolve errors for resource quotas](#).

General limits

For limits on resource names, see [Naming rules and restrictions for Azure resources](#).

For information about Resource Manager API read and write limits, see [Throttling Resource Manager requests](#).

Management group limits

The following limits apply to [management groups](#).

RESOURCE	LIMIT
Management groups per Azure AD tenant	10,000
Subscriptions per management group	Unlimited.
Levels of management group hierarchy	Root level plus 6 levels ¹
Direct parent management group per management group	One
Management group level deployments per location	800 ²
Locations of Management group level deployments	10

¹The 6 levels don't include the subscription level.

²If you reach the limit of 800 deployments, delete deployments from the history that are no longer needed. To delete management group level deployments, use [Remove-AzManagementGroupDeployment](#) or [az deployment mg delete](#).

Subscription limits

The following limits apply when you use Azure Resource Manager and Azure resource groups.

RESOURCE	LIMIT
Subscriptions associated with an Azure Active Directory tenant	Unlimited
Coadministrators per subscription	Unlimited
Resource groups per subscription	980
Azure Resource Manager API request size	4,194,304 bytes
Tags per subscription ¹	50
Unique tag calculations per subscription ¹	80,000
Subscription-level deployments per location	800 ²
Locations of Subscription-level deployments	10

¹You can apply up to 50 tags directly to a subscription. However, the subscription can contain an unlimited number of tags that are applied to resource groups and resources within the subscription. The number of tags per resource or resource group is limited to 50. Resource Manager returns a [list of unique tag name and values](#) in the subscription only when the number of tags is 80,000 or less. You still can find a resource by tag when the number exceeds 80,000.

²Deployments are automatically deleted from the history as you near the limit. For more information, see [Automatic deletions from deployment history](#).

Resource group limits

RESOURCE	LIMIT
Resources per resource group	Resources aren't limited by resource group. Instead, they're limited by resource type in a resource group. See next row.
Resources per resource group, per resource type	800 - Some resource types can exceed the 800 limit. See Resources not limited to 800 instances per resource group .
Deployments per resource group in the deployment history	800 ¹
Resources per deployment	800
Management locks per unique scope	20
Number of tags per resource or resource group	50
Tag key length	512
Tag value length	256

¹Deployments are automatically deleted from the history as you near the limit. Deleting an entry from the deployment history doesn't affect the deployed resources. For more information, see [Automatic deletions from deployment history](#).

Template limits

VALUE	LIMIT
Parameters	256
Variables	256
Resources (including copy count)	800
Outputs	64
Template expression	24,576 chars
Resources in exported templates	200
Template size	4 MB
Parameter file size	4 MB

You can exceed some template limits by using a nested template. For more information, see [Use linked templates when you deploy Azure resources](#). To reduce the number of parameters, variables, or outputs, you can combine several values into an object. For more information, see [Objects as parameters](#).

You may get an error with a template or parameter file of less than 4 MB, if the total size of the request is too large. For more information about how to simplify your template to avoid a large request, see [Resolve errors for job size exceeded](#).

Active Directory limits

Here are the usage constraints and other service limits for the Azure Active Directory (Azure AD) service.

CATEGORY	LIMIT
Tenants	<p>A single user can belong to a maximum of 500 Azure AD tenants as a member or a guest.</p> <p>A single user can create a maximum of 200 directories.</p>
Domains	<p>You can add no more than 5000 managed domain names. If you set up all of your domains for federation with on-premises Active Directory, you can add no more than 2500 domain names in each tenant.</p>
Resources	<ul style="list-style-type: none">• A maximum of 50,000 Azure AD resources can be created in a single tenant by users of the Free edition of Azure Active Directory by default. If you have at least one verified domain, the default Azure AD service quota for your organization is extended to 300,000 Azure AD resources. Azure AD service quota for organizations created by self-service sign-up remains 50,000 Azure AD resources even after you performed an internal admin takeover and the organization is converted to a managed tenant with at least one verified domain. This service limit is unrelated to the pricing tier limit of 500,000 resources on the Azure AD pricing page. To go beyond the default quota, you must contact Microsoft Support.• A non-admin user can create no more than 250 Azure AD resources. Both active resources and deleted resources that are available to restore count toward this quota. Only deleted Azure AD resources that were deleted fewer than 30 days ago are available to restore. Deleted Azure AD resources that are no longer available to restore count toward this quota at a value of one-quarter for 30 days. If you have developers who are likely to repeatedly exceed this quota in the course of their regular duties, you can create and assign a custom role with permission to create a limitless number of app registrations.
Schema extensions	<ul style="list-style-type: none">• String-type extensions can have a maximum of 256 characters.• Binary-type extensions are limited to 256 bytes.• Only 100 extension values, across <i>all</i> types and <i>all</i> applications, can be written to any single Azure AD resource.• Only User, Group, TenantDetail, Device, Application, and ServicePrincipal entities can be extended with string-type or binary-type single-valued attributes.

CATEGORY	LIMIT
Applications	<ul style="list-style-type: none">• A maximum of 100 users can be owners of a single application.• A user, group, or service principal can have a maximum of 1,500 app role assignments.• Password-based single sign-on (SSO) app has a limit of 48 users, which means that there is a limit of 48 keys for username/password pairs per app. If you want to add additional users, see the troubleshooting instructions in Troubleshoot password-based single sign-on in Azure AD.• A user can only have a maximum of 48 apps where they have username and password credentials configured.
Application Manifest	A maximum of 1200 entries can be added in the Application Manifest.

CATEGORY	LIMIT
Groups	<ul style="list-style-type: none"> • A non-admin user can create a maximum of 250 groups in an Azure AD organization. Any Azure AD admin who can manage groups in the organization can also create unlimited number of groups (up to the Azure AD object limit). If you assign a role to remove the limit for a user, assign them to a less privileged built-in role such as User Administrator or Groups Administrator. • An Azure AD organization can have a maximum of 5000 dynamic groups. • A maximum of 300 role-assignable groups can be created in a single Azure AD organization (tenant). • A maximum of 100 users can be owners of a single group. • Any number of Azure AD resources can be members of a single group. • A user can be a member of any number of groups. • By default, the number of members in a group that you can synchronize from your on-premises Active Directory to Azure Active Directory by using Azure AD Connect is limited to 50,000 members. If you need to synch a group membership that's over this limit, you must onboard the Azure AD Connect Sync V2 endpoint API. • Nested Groups in Azure AD are not supported within all scenarios • Group expiration policy can be assigned to a maximum of 500 Microsoft 365 groups, when selecting a list of groups. There is no limit when the policy is applied to all Microsoft 365 groups. <p>At this time the following are the supported scenarios with nested groups.</p> <ul style="list-style-type: none"> • One group can be added as a member of another group and you can achieve group nesting. • Group membership claims (when an app is configured to receive group membership claims in the token, nested groups in which the signed-in user is a member are included) • Conditional access (when a conditional access policy has a group scope) • Restricting access to self-serve password reset • Restricting which users can do Azure AD Join and device registration <p>The following scenarios DO NOT supported nested groups:</p> <ul style="list-style-type: none"> • App role assignment (assigning groups to an app is supported, but groups nested within the directly assigned group will not have access), both for access and for provisioning • Group-based licensing (assigning a license automatically to all members of a group) • Microsoft 365 Groups.

CATEGORY	LIMIT
Application Proxy	<ul style="list-style-type: none"> • A maximum of 500 transactions per second per App Proxy application • A maximum of 750 transactions per second for the Azure AD organization <p>A transaction is defined as a single http request and response for a unique resource. When throttled, clients will receive a 429 response (too many requests).</p>
Access Panel	There's no limit to the number of applications that can be seen in the Access Panel per user regardless of assigned licenses.
Reports	A maximum of 1,000 rows can be viewed or downloaded in any report. Any additional data is truncated.
Administrative units	An Azure AD resource can be a member of no more than 30 administrative units.
Azure AD roles and permissions	<ul style="list-style-type: none"> • A maximum of 30 Azure AD custom roles can be created in an Azure AD organization. • A group can't be added as a group owner. • A user's ability to read other users' tenant information can be restricted only by the Azure AD organization-wide switch to disable all non-admin users' access to all tenant information (not recommended). For more information, see To restrict the default permissions for member users. • It may take up to 15 minutes or signing out/signing in before admin role membership additions and revocations take effect.

API Management limits

RESOURCE	LIMIT
Maximum number of scale units	12 per region ¹
Cache size	5 GiB per unit ²
Concurrent back-end connections ³ per HTTP authority	2,048 per unit ⁴
Maximum cached response size	2 MiB
Maximum policy document size	256 KiB ⁵
Maximum custom gateway domains per service instance ⁶	20
Maximum number of CA certificates per service instance ⁷	10
Maximum number of service instances per subscription ⁸	20

RESOURCE	LIMIT
Maximum number of subscriptions per service instance ⁸	500
Maximum number of client certificates per service instance ⁸	50
Maximum number of APIs per service instance ⁸	50
Maximum number of API management operations per service instance ⁸	1,000
Maximum total request duration ⁸	30 seconds
Maximum buffered payload size ⁸	2 MiB
Maximum request URL size ⁹	4096 bytes
Maximum length of URL path segment ¹⁰	260 characters
Maximum size of API schema used by validation policy ¹⁰	4 MB
Maximum size of request or response body in validate-content policy	100 KB
Maximum number of self-hosted gateways ¹¹	25

¹Scaling limits depend on the pricing tier. For details on the pricing tiers and their scaling limits, see [API Management pricing](#).

²Per unit cache size depends on the pricing tier. To see the pricing tiers and their scaling limits, see [API Management pricing](#).

³Connections are pooled and reused unless explicitly closed by the back end.

⁴This limit is per unit of the Basic, Standard, and Premium tiers. The Developer tier is limited to 1,024. This limit doesn't apply to the Consumption tier.

⁵This limit applies to the Basic, Standard, and Premium tiers. In the Consumption tier, policy document size is limited to 16 KiB.

⁶Multiple custom domains are supported in the Developer and Premium tiers only.

⁷CA certificates are not supported in the Consumption tier.

⁸This limit applies to the Consumption tier only. There are no limits in these categories for other tiers.

⁹Applies to the Consumption tier only. Includes an up to 2048 bytes long query string.

¹⁰ To increase this limit, please contact [support](#).

¹¹Self-hosted gateways are supported in the Developer and Premium tiers only. The limit applies to the number of [self-hosted gateway resources](#). To raise this limit please contact [support](#). Note, that the number of nodes (or replicas) associated with a self-hosted gateway resource is unlimited in the Premium tier and capped at a single node in the Developer tier.

App Service limits

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V1-V3)	ISOLATED
Web, mobile, or API apps per Azure App Service plan ¹	10	100	Unlimited ²	Unlimited ²	Unlimited ²	Unlimited ²
App Service plan	10 per region	10 per resource group	100 per resource group	100 per resource group	100 per resource group	100 per resource group
Compute instance type	Shared	Shared	Dedicated ³	Dedicated ³	Dedicated ³	Dedicated ³
Scale out (maximum instances)	1 shared	1 shared	3 dedicated ³	10 dedicated ³	20 dedicated for v1; 30 dedicated for v2 and v3. ³	100 dedicated ⁴
Storage ⁵	1 GB ⁵	1 GB ⁵	10 GB ⁵	50 GB ⁵	250 GB ⁵	1 TB ⁵ The available storage quota is 999 GB.
CPU time (5 minutes) ⁶	3 minutes	3 minutes	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates
CPU time (day) ⁶	60 minutes	240 minutes	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates
Memory (1 hour)	1,024 MB per App Service plan	1,024 MB per app	N/A	N/A	N/A	N/A
Bandwidth	165 MB	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply
Application architecture	32-bit	32-bit	32-bit/64-bit	32-bit/64-bit	32-bit/64-bit	32-bit/64-bit
Web sockets per instance ⁷	5	35	350	Unlimited	Unlimited	Unlimited
Outbound IP connections per instance	600	600	Depends on instance size ⁸	Depends on instance size ⁸	Depends on instance size ⁸	16,000

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V1-V3)	ISOLATED
Concurrent debugger connections per application	1	1	1	5	5	5
App Service Certificates per subscription ⁹	Not supported	Not supported	10	10	10	10
Custom domains per app	0 (azurewebsite.s.net subdomain only)	500	500	500	500	500
Custom domain SSL support	Not supported, wildcard certificate for *.azurewebsites.net available by default	Not supported, wildcard certificate for *.azurewebsites.net available by default	Unlimited SNI SSL connections	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included
Hybrid connections			5 per plan	25 per plan	220 per app	220 per app
Virtual Network Integration				X	X	X
Private Endpoints					100 per app	
Integrated load balancer		X	X	X	X	X ¹⁰
Access restrictions	512 rules per app	512 rules per app	512 rules per app	512 rules per app	512 rules per app	512 rules per app
Always On			X	X	X	X
Scheduled backups				Scheduled backups every 2 hours, a maximum of 12 backups per day (manual + scheduled)	Scheduled backups every hour, a maximum of 50 backups per day (manual + scheduled)	Scheduled backups every hour, a maximum of 50 backups per day (manual + scheduled)
Autoscale				X	X	X

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V1-V3)	ISOLATED
WebJobs ¹¹	X	X	X	X	X	X
Endpoint monitoring			X	X	X	X
Staging slots per app				5	20	20
Testing in Production				X	X	X
Diagnostic Logs	X	X	X	X	X	X
Kudu	X	X	X	X	X	X
Authentication and Authorization	X	X	X	X	X	X
App Service Managed Certificates (Public Preview) ¹²			X	X	X	X
SLA			99.95%	99.95%	99.95%	99.95%

¹ Apps and storage quotas are per App Service plan unless noted otherwise.

² The actual number of apps that you can host on these machines depends on the activity of the apps, the size of the machine instances, and the corresponding resource utilization.

³ Dedicated instances can be of different sizes. For more information, see [App Service pricing](#).

⁴ More are allowed upon request.

⁵ The storage limit is the total content size across all apps in the same App service plan. The total content size of all apps across all App service plans in a single resource group and region cannot exceed 500 GB. The file system quota for App Service hosted apps is determined by the aggregate of App Service plans created in a region and resource group.

⁶ These resources are constrained by physical resources on the dedicated instances (the instance size and the number of instances).

⁷ If you scale an app in the Basic tier to two instances, you have 350 concurrent connections for each of the two instances. For Standard tier and above, there are no theoretical limits to web sockets, but other factors can limit the number of web sockets. For example, maximum concurrent requests allowed (defined by `maxConcurrentRequestsPerCpu`) are: 7,500 per small VM, 15,000 per medium VM (7,500 x 2 cores), and 75,000 per large VM (18,750 x 4 cores).

⁸ The maximum IP connections are per instance and depend on the instance size: 1,920 per B1/S1/P1V3 instance, 3,968 per B2/S2/P2V3 instance, 8,064 per B3/S3/P3V3 instance.

⁹ The App Service Certificate quota limit per subscription can be increased via a support request to a maximum

limit of 200.

¹⁰ App Service Isolated SKUs can be internally load balanced (ILB) with Azure Load Balancer, so there's no public connectivity from the internet. As a result, some features of an ILB Isolated App Service must be used from machines that have direct access to the ILB network endpoint.

¹¹ Run custom executables and/or scripts on demand, on a schedule, or continuously as a background task within your App Service instance. Always On is required for continuous WebJobs execution. There's no predefined limit on the number of WebJobs that can run in an App Service instance. There are practical limits that depend on what the application code is trying to do.

¹² Naked domains aren't supported. Only issuing standard certificates (wildcard certificates aren't available). Limited to only one free certificate per custom domain.

Automation limits

Process automation

RESOURCE	LIMIT	NOTES
Maximum number of new jobs that can be submitted every 30 seconds per Azure Automation account (nonscheduled jobs)	100	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response.
Maximum number of concurrent running jobs at the same instance of time per Automation account (nonscheduled jobs)	200	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response.
Maximum storage size of job metadata for a 30-day rolling period	10 GB (approximately 4 million jobs)	When this limit is reached, the subsequent requests to create a job fail.
Maximum job stream limit	1 MiB	A single stream cannot be larger than 1 MiB.
Maximum number of modules that can be imported every 30 seconds per Automation account	5	
Maximum size of a module	100 MB	
Maximum size of a node configuration file	1 MB	Applies to state configuration
Job run time, Free tier	500 minutes per subscription per calendar month	
Maximum amount of disk space allowed per sandbox ¹	1 GB	Applies to Azure sandboxes only.
Maximum amount of memory given to a sandbox ¹	400 MB	Applies to Azure sandboxes only.
Maximum number of network sockets allowed per sandbox ¹	1,000	Applies to Azure sandboxes only.

RESOURCE	LIMIT	NOTES
Maximum runtime allowed per runbook ¹	3 hours	Applies to Azure sandboxes only.
Maximum number of Automation accounts in a subscription	No limit	
Maximum number of Hybrid Worker Groups per Automation Account	4,000	
Maximum number of concurrent jobs that can be run on a single Hybrid Runbook Worker	50	
Maximum runbook job parameter size	512 kilobytes	
Maximum runbook parameters	50	If you reach the 50-parameter limit, you can pass a JSON or XML string to a parameter and parse it with the runbook.
Maximum webhook payload size	512 kilobytes	
Maximum days that job data is retained	30 days	
Maximum PowerShell workflow state size	5 MB	Applies to PowerShell workflow runbooks when checkpointing workflow.

¹A sandbox is a shared environment that can be used by multiple jobs. Jobs that use the same sandbox are bound by the resource limitations of the sandbox.

Change Tracking and Inventory

The following table shows the tracked item limits per machine for change tracking.

RESOURCE	LIMIT	NOTES
File	500	
File size	5 MB	
Registry	250	
Windows software	250	Doesn't include software updates.
Linux packages	1,250	
Services	250	
Daemon	250	

Update Management

The following table shows the limits for Update Management.

RESOURCE	LIMIT	NOTES
Number of machines per update deployment	1000	
Number of dynamic groups per update deployment	500	

Azure App Configuration

RESOURCE	LIMIT
Configuration stores - Free tier	1 per subscription
Configuration stores - Standard tier	unlimited per subscription
Configuration store requests - Free tier	1,000 requests per day
Configuration store requests - Standard tier	Throttling starts at 20,000 requests per hour
Storage - Free tier	10 MB
Storage - Standard tier	1 GB
keys and values	10 KB for a single key-value item

Azure API for FHIR service limits

Azure API for FHIR is a managed, standards-based, compliant API for clinical health data that enables solutions for actionable analytics and machine learning.

QUOTA NAME	DEFAULT LIMIT	MAXIMUM LIMIT	NOTES
Request Units (RUs)	10,000 RUs	Contact support Maximum available is 1,000,000.	You need a minimum of 400 RUs or 40 RUs/GB, whichever is larger.
Concurrent connections	15 concurrent connections on two instances (for a total of 30 concurrent requests)	Contact support	
Azure API for FHIR Service Instances per Subscription	10	Contact support	

Azure Cache for Redis limits

RESOURCE	LIMIT
Cache size	1.2 TB
Databases	64

RESOURCE	LIMIT
Maximum connected clients	40,000
Azure Cache for Redis replicas, for high availability	1
Shards in a premium cache with clustering	10

Azure Cache for Redis limits and sizes are different for each pricing tier. To see the pricing tiers and their associated sizes, see [Azure Cache for Redis pricing](#).

For more information on Azure Cache for Redis configuration limits, see [Default Redis server configuration](#).

Because configuration and management of Azure Cache for Redis instances is done by Microsoft, not all Redis commands are supported in Azure Cache for Redis. For more information, see [Redis commands not supported in Azure Cache for Redis](#).

Azure Cloud Services limits

RESOURCE	LIMIT
Web or worker roles per deployment ¹	25
Instance input endpoints per deployment	25
Input endpoints per deployment	25
Internal endpoints per deployment	25
Hosted service certificates per deployment	199

¹Each Azure Cloud Service with web or worker roles can have two deployments, one for production and one for staging. This limit refers to the number of distinct roles, that is, configuration. This limit doesn't refer to the number of instances per role, that is, scaling.

Azure Cognitive Search limits

Pricing tiers determine the capacity and limits of your search service. Tiers include:

- **Free** multi-tenant service, shared with other Azure subscribers, is intended for evaluation and small development projects.
- **Basic** provides dedicated computing resources for production workloads at a smaller scale, with up to three replicas for highly available query workloads.
- **Standard**, which includes S1, S2, S3, and S3 High Density, is for larger production workloads. Multiple levels exist within the Standard tier so that you can choose a resource configuration that best matches your workload profile.

Limits per subscription

You can create multiple services within a subscription. Each one can be provisioned at a specific tier. You're limited only by the number of services allowed at each tier. For example, you could create up to 12 services at the Basic tier and another 12 services at the S1 tier within the same subscription. For more information about tiers, see [Choose an SKU or tier for Azure Cognitive Search](#).

Maximum service limits can be raised upon request. If you need more services within the same subscription, contact Azure Support.

RESOURCE	FREE ¹	BASIC	S1	S2	S3	S3 HD	L1	L2
Maximum services	1	16	16	8	6	6	6	6
Maximum scale in search units (SU) ²	N/A	3 SU	36 SU	36 SU	36 SU	36 SU	36 SU	36 SU

¹ Free is based on shared, not dedicated, resources. Scale-up is not supported on shared resources.

² Search units are billing units, allocated as either a *replica* or a *partition*. You need both resources for storage, indexing, and query operations. To learn more about SU computations, see [Scale resource levels for query and index workloads](#).

Limits per search service

A search service is constrained by disk space or by a hard limit on the maximum number of indexes or indexers, whichever comes first. The following table documents storage limits. For maximum object limits, see [Limits by resource](#).

RESOURCE	FREE	BASIC ¹	S1	S2	S3	S3 HD	L1	L2
Service level agreement (SLA) ²	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Storage per partition	50 MB	2 GB	25 GB	100 GB	200 GB	200 GB	1 TB	2 TB
Partitions per service	N/A	1	12	12	12	3	12	12
Partition size	N/A	2 GB	25 GB	100 GB	200 GB	200 GB	1 TB	2 TB
Replicas	N/A	3	12	12	12	12	12	12

¹ Basic has one fixed partition. Additional search units can be used to add replicas for larger query volumes.

² Service level agreements are in effect for billable services on dedicated resources. Free services and preview features have no SLA. For billable services, SLAs take effect when you provision sufficient redundancy for your service. Two or more replicas are required for query (read) SLAs. Three or more replicas are required for query and indexing (read-write) SLAs. The number of partitions isn't an SLA consideration.

To learn more about limits on a more granular level, such as document size, queries per second, keys, requests, and responses, see [Service limits in Azure Cognitive Search](#).

Azure Cognitive Services limits

The following limits are for the number of Cognitive Services resources per Azure subscription. Each of the Cognitive Services may have additional limitations, for more information see [Azure Cognitive Services](#).

TYPE	LIMIT	EXAMPLE
A mixture of Cognitive Services resources	Maximum of 200 total Cognitive Services resources.	100 Computer Vision resources in West US 2, 50 Speech Service resources in West US, and 50 Text Analytics resources in East US.
A single type of Cognitive Services resources.	Maximum of 100 resources per region, with a maximum of 200 total Cognitive Services resources.	100 Computer Vision resources in West US 2, and 100 Computer Vision resources in East US.

Azure Cosmos DB limits

For Azure Cosmos DB limits, see [Limits in Azure Cosmos DB](#).

Azure Data Explorer limits

The following table describes the maximum limits for Azure Data Explorer clusters.

RESOURCE	LIMIT
Clusters per region per subscription	20
Instances per cluster	1000
Number of databases in a cluster	10,000
Number of attached database configurations in a cluster	70

The following table describes the limits on management operations performed on Azure Data Explorer clusters.

SCOPE	OPERATION	LIMIT
Cluster	read (for example, get a cluster)	500 per 5 minutes
Cluster	write (for example, create a database)	1000 per hour

Azure Database for MySQL

For Azure Database for MySQL limits, see [Limitations in Azure Database for MySQL](#).

Azure Database for PostgreSQL

For Azure Database for PostgreSQL limits, see [Limitations in Azure Database for PostgreSQL](#).

Azure Functions limits

RESOURCE	CONSUMPTION PLAN	PREMIUM PLAN	DEDICATED PLAN	ASE	KUBERNETES
Default timeout duration (min)	5	30	30 ¹	30	30
Max timeout duration (min)	10	unbounded ⁷	unbounded ²	unbounded	unbounded
Max outbound connections (per instance)	600 active (1200 total)	unbounded	unbounded	unbounded	unbounded
Max request size (MB) ³	100	100	100	100	Depends on cluster
Max query string length ³	4096	4096	4096	4096	Depends on cluster
Max request URL length ³	8192	8192	8192	8192	Depends on cluster
ACU per instance	100	210-840	100-840	210-250 ⁸	AKS pricing
Max memory (GB per instance)	1.5	3.5-14	1.75-14	3.5 - 14	Any node is supported
Max instance count	200	100 ⁹	varies by SKU ¹⁰	100 ¹⁰	Depends on cluster
Function apps per plan	100	100	unbounded ⁴	unbounded	unbounded
App Service plans	100 per region	100 per resource group	100 per resource group	-	-
Storage ⁵	5 TB	250 GB	50-1000 GB	1 TB	n/a
Custom domains per app	500 ⁶	500	500	500	n/a
Custom domain SSL support	unbounded SNI SSL connection included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included	n/a

¹ By default, the timeout for the Functions 1.x runtime in an App Service plan is unbounded.

² Requires the App Service plan be set to [Always On](#). Pay at standard [rates](#).

³ These limits are [set in the host](#).

⁴ The actual number of function apps that you can host depends on the activity of the apps, the size of the machine instances, and the corresponding resource utilization.

⁵ The storage limit is the total content size in temporary storage across all apps in the same App Service plan. Consumption plan uses Azure Files for temporary storage.

⁶ When your function app is hosted in a [Consumption plan](#), only the CNAME option is supported. For function apps in a [Premium plan](#) or an [App Service plan](#), you can map a custom domain using either a CNAME or an A

record.

⁷ Guaranteed for up to 60 minutes.

⁸ 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.

⁹ When running on Linux in a Premium plan, you're currently limited to 20 instances.

¹⁰ See [App Service limits](#) for details.

For more information, see [Functions Hosting plans comparison](#).

Azure Kubernetes Service limits

RESOURCE	LIMIT
Maximum clusters per subscription	5000
Maximum nodes per cluster with Virtual Machine Availability Sets and Basic Load Balancer SKU	100
Maximum nodes per cluster with Virtual Machine Scale Sets and Standard Load Balancer SKU	1000 (across all node pools)
Maximum node pools per cluster	100
Maximum pods per node: Basic networking with Kubenet	Maximum: 250 Azure CLI default: 110 Azure Resource Manager template default: 110 Azure portal deployment default: 30
Maximum pods per node: Advanced networking with Azure Container Networking Interface	Maximum: 250 Default: 30
Open Service Mesh (OSM) AKS addon preview	Kubernetes Cluster Version: 1.19+ ¹ OSM controllers per cluster: 1 ¹ Pods per OSM controller: 500 ¹ Kubernetes service accounts managed by OSM: 50 ¹

¹The OSM add-on for AKS is in a preview state and will undergo additional enhancements before general availability (GA). During the preview phase, it's recommended to not surpass the limits shown.

Azure Machine Learning limits

The latest values for Azure Machine Learning Compute quotas can be found in the [Azure Machine Learning quota page](#)

Azure Maps limits

The following table shows the usage limit for the Azure Maps S0 pricing tier. Usage limit depends on the pricing tier.

RESOURCE	S0 PRICING TIER LIMIT
Maximum request rate per subscription	50 requests per second

The following table shows the cumulative data size limit for Azure Maps accounts in an Azure subscription. The Azure Maps Data service is available only at the S1 pricing tier.

RESOURCE	LIMIT
Maximum storage per Azure subscription	1 GB
Maximum size per file upload	100 MB

For more information on the Azure Maps pricing tiers, see [Azure Maps pricing](#).

Azure Monitor limits

Alerts

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Metric alerts (classic)	100 active alert rules per subscription.	Call support
Metric alerts	5,000 active alert rules per subscription in Azure public, Azure China 21Vianet and Azure Government clouds. If you are hitting this limit, explore if you can use same type multi-resource alerts . 5,000 metric time-series per alert rule.	Call support.
Activity log alerts	100 active alert rules per subscription (cannot be increased).	Same as default
Log alerts	1000 active alert rules per subscription. 1000 active alert rules per resource.	Call support
Alert rules and Action rules description length	Log search alerts 4096 characters All other 2048 characters	Same as default

Alerts API

Azure Monitor Alerts have several throttling limits to protect against users making an excessive number of calls. Such behavior can potentially overload the system backend resources and jeopardize service responsiveness. The following limits are designed to protect customers from interruptions and ensure consistent service level. The user throttling and limits are designed to impact only extreme usage scenario and should not be relevant for typical usage.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
GET alertsSummary	50 calls per minute per subscription	Same as default
GET alerts (without specifying an alert ID)	100 calls per minute per subscription	Same as default
All other calls	1000 calls per minute per subscription	Same as default

Action groups

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
----------	---------------	---------------

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure app push	10 Azure app actions per action group.	Same as Default
Email	1,000 email actions in an action group. No more than 100 emails in an hour. Also see the rate limiting information .	Same as Default
ITSM	10 ITSM actions in an action group.	Same as Default
Logic app	10 logic app actions in an action group.	Same as Default
Runbook	10 runbook actions in an action group.	Same as Default
SMS	10 SMS actions in an action group. No more than 1 SMS message every 5 minutes. Also see the rate limiting information .	Same as Default
Voice	10 voice actions in an action group. No more than 1 voice call every 5 minutes. Also see the rate limiting information .	Same as Default
Webhook	10 webhook actions in an action group. Maximum number of webhook calls is 1500 per minute per subscription. Other limits are available at action-specific information .	Same as Default

Autoscale

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Autoscale settings	100 per region per subscription.	Same as default
Autoscale profiles	20 profiles per autoscale setting.	Same as default

Log queries and language

General query limits

LIMIT	DESCRIPTION
Query language	Azure Monitor uses the same Kusto query language as Azure Data Explorer. See Azure Monitor log query language differences for KQL language elements not supported in Azure Monitor.
Azure regions	Log queries can experience excessive overhead when data spans Log Analytics workspaces in multiple Azure regions. See Query limits for details.

LIMIT	DESCRIPTION
Cross resource queries	Maximum number of Application Insights resources and Log Analytics workspaces in a single query limited to 100. Cross-resource query is not supported in View Designer. Cross-resource query in log alerts is supported in the new <code>scheduledQueryRules</code> API. See Cross-resource query limits for details.

User query throttling

Azure Monitor has several throttling limits to protect against users sending an excessive number of queries. Such behavior can potentially overload the system backend resources and jeopardize service responsiveness. The following limits are designed to protect customers from interruptions and ensure consistent service level. The user throttling and limits are designed to impact only extreme usage scenario and should not be relevant for typical usage.

MEASURE	LIMIT PER USER	DESCRIPTION
Concurrent queries	5	If there are already 5 queries running for the user, any new queries are placed in a per-user concurrency queue. When one of the running queries ends, the next query will be pulled from the queue and started. This does not include queries from alert rules.
Time in concurrency queue	3 minutes	If a query sits in the queue for more than 3 minutes without being started, it will be terminated with an HTTP error response with code 429.
Total queries in concurrency queue	200	Once the number of queries in the queue reaches 200, any additional queries will be rejected with an HTTP error code 429. This number is in addition to the 5 queries that can be running simultaneously.
Query rate	200 queries per 30 seconds	This is the overall rate that queries can be submitted by a single user to all workspaces. This limit applies to programmatic queries or queries initiated by visualization parts such as Azure dashboards and the Log Analytics workspace summary page.

- Optimize your queries as described in [Optimize log queries in Azure Monitor](#).
- Dashboards and workbooks can contain multiple queries in a single view that generate a burst of queries every time they load or refresh. Consider breaking them up into multiple views that load on demand.
- In Power BI, consider extracting only aggregated results rather than raw logs.

Log Analytics workspaces

Data collection volume and retention

TIER	LIMIT PER DAY	DATA RETENTION	COMMENT
Current Per GB pricing tier (introduced April 2018)	No limit	30 - 730 days	Data retention beyond 31 days is available for additional charges. Learn more about Azure Monitor pricing.
Legacy Free tiers (introduced April 2016)	500 MB	7 days	When your workspace reaches the 500 MB per day limit, data ingestion stops and resumes at the start of the next day. A day is based on UTC. Note that data collected by Azure Security Center is not included in this 500 MB per day limit and will continue to be collected above this limit.
Legacy Standalone Per GB tier (introduced April 2016)	No limit	30 to 730 days	Data retention beyond 31 days is available for additional charges. Learn more about Azure Monitor pricing.
Legacy Per Node (OMS) (introduced April 2016)	No limit	30 to 730 days	Data retention beyond 31 days is available for additional charges. Learn more about Azure Monitor pricing.
Legacy Standard tier	No limit	30 days	Retention can't be adjusted
Legacy Premium tier	No limit	365 days	Retention can't be adjusted

Number of workspaces per subscription.

PRICING TIER	WORKSPACE LIMIT	COMMENTS
Free tier	10	This limit can't be increased.
All other tiers	No limit	You're limited by the number of resources within a resource group and the number of resource groups per subscription.

Azure portal

CATEGORY	LIMIT	COMMENTS
Maximum records returned by a log query	30,000	Reduce results using query scope, time range, and filters in the query.

Data Collector API

CATEGORY	LIMIT	COMMENTS
Maximum size for a single post	30 MB	Split larger volumes into multiple posts.
Maximum size for field values	32 KB	Fields longer than 32 KB are truncated.

Search API

CATEGORY	LIMIT	COMMENTS
Maximum records returned in a single query	500,000	
Maximum size of data returned	~104 MB (~100 MiB)	
Maximum query running time	10 minutes	See Timeouts for details.
Maximum request rate	200 requests per 30 seconds per Azure AD user or client IP address	See Rate limits for details.

Azure Monitor Logs connector

CATEGORY	LIMIT	COMMENTS
Max number of records	500,000	
Maximum size of data returned	~104 MB (~100 MiB)	
Max query timeout	110 second	
Charts		Visualization in Logs page and the connector are using different charting libraries and some functionality isn't available in the connector currently.

General workspace limits

CATEGORY	LIMIT	COMMENTS
Maximum columns in a table	500	
Maximum characters for column name	500	

Data ingestion volume rate

Azure Monitor is a high scale data service that serves thousands of customers sending terabytes of data each month at a growing pace. The volume rate limit intends to isolate Azure Monitor customers from sudden ingestion spikes in multitenancy environment. A default ingestion volume rate threshold of 500 MB (compressed) is defined in workspaces, this is translated to approximately **6 GB/min** uncompressed -- the actual size can vary between data types depending on the log length and its compression ratio. The volume rate limit applies to data ingested from Azure resources via [Diagnostic settings](#). When volume rate limit is reached, a retry mechanism attempts to ingest the data 4 times in a period of 30 minutes and drop it if operation fails. It doesn't apply to data ingested from [agents](#) or [Data Collector API](#).

When data sent to your workspace is at a volume rate higher than 80% of the threshold configured in your workspace, an event is sent to the *Operation* table in your workspace every 6 hours while the threshold continues to be exceeded. When ingested volume rate is higher than threshold, some data is dropped and an event is sent to the *Operation* table in your workspace every 6 hours while the threshold continues to be exceeded. If your ingestion volume rate continues to exceed the threshold or you are expecting to reach it sometime soon, you can request to increase it in by opening a support request.

See [Monitor health of Log Analytics workspace in Azure Monitor](#) to create alert rules to be proactively notified when you reach any ingestion limits.

NOTE

Depending on how long you've been using Log Analytics, you might have access to legacy pricing tiers. Learn more about [Log Analytics legacy pricing tiers](#).

Application Insights

There are some limits on the number of metrics and events per application, that is, per instrumentation key. Limits depend on the [pricing plan](#) that you choose.

RESOURCE	DEFAULT LIMIT	NOTE
Total data per day	100 GB	You can reduce data by setting a cap. If you need more data, you can increase the limit in the portal, up to 1,000 GB. For capacities greater than 1,000 GB, send email to AIDataCap@microsoft.com .
Throttling	32,000 events/second	The limit is measured over a minute.
Data retention Logs	30 - 730 days	This resource is for Logs .
Data retention Metrics	90 days	This resource is for Metrics Explorer .
Availability multi-step test detailed results retention	90 days	This resource provides detailed results of each step.
Maximum telemetry item size	64 kB	
Maximum telemetry items per batch	64 K	
Property and metric name length	150	See type schemas .
Property value string length	8,192	See type schemas .
Trace and exception message length	32,768	See type schemas .
Availability tests count per app	100	
Profiler data retention	5 days	
Profiler data sent per day	10 GB	

For more information, see [About pricing and quotas in Application Insights](#).

Azure Policy limits

There's a maximum count for each object type for Azure Policy. For definitions, an entry of *Scope* means the [management group](#) or subscription. For assignments and exemptions, an entry of *Scope* means the [management group](#), subscription, resource group, or individual resource.

WHERE	WHAT	MAXIMUM COUNT
Scope	Policy definitions	500
Scope	Initiative definitions	200
Tenant	Initiative definitions	2,500
Scope	Policy or initiative assignments	200
Scope	Exemptions	1000
Policy definition	Parameters	20
Initiative definition	Policies	1000
Initiative definition	Parameters	100
Policy or initiative assignments	Exclusions (notScopes)	400
Policy rule	Nested conditionals	512
Remediation task	Resources	500

Azure Quantum limits

Provider Limits & Quota

The Azure Quantum Service supports both first and third-party service providers. Third-party providers own their limits and quotas. Users can view offers and limits in the Azure portal when configuring third-party providers in the provider blade.

You can find the published quota limits for Microsoft's first party Optimization Solutions provider below.

Learn & Develop SKU

RESOURCE	LIMIT
CPU-based concurrent jobs	up to 5 concurrent jobs
FPGA-based concurrent jobs	up to 2 concurrent jobs
CPU-based solver hours	20 hours per month
FPGA-based solver hours	1 hour per month

If you are using the Learn & Develop SKU, you **cannot** request an increase on your quota limits. Instead you should switch to the Performance at Scale SKU.

Performance at Scale SKU

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
CPU-based concurrent jobs	up to 100 concurrent jobs	same as default limit
FPGA-based concurrent jobs	up to 10 concurrent jobs	same as default limit
Solver hours	1,000 hours per month	up to 50,000 hours per month

If you need to request a limit increase, please reach out to Azure Support.

For more information, please review the [Azure Quantum pricing page](#). For information on third-party offerings, please review the relevant provider page in the Azure portal.

Azure RBAC limits

The following limits apply to [Azure role-based access control \(Azure RBAC\)](#).

RESOURCE	LIMIT
Azure role assignments per Azure subscription	2,000
Azure role assignments per management group	500
Size of description for Azure role assignments	2 KB
Size of condition for Azure role assignments	8 KB
Azure custom roles per tenant	5,000
Azure custom roles per tenant (for Azure Germany and Azure China 21Vianet)	2,000

Azure SignalR Service limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure SignalR Service units per instance for Free tier	1	1
Azure SignalR Service units per instance for Standard tier	100	100
Azure SignalR Service units per subscription per region for Free tier	5	5
Total Azure SignalR Service unit counts per subscription per region	150	Unlimited
Connections per unit per day for Free tier	20	20

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Connections per unit per day for Standard tier	1,000	1,000
Included messages per unit per day for Free tier	20,000	20,000
Additional messages per unit per day for Free tier	0	0
Included messages per unit per day for Standard tier	1,000,000	1,000,000
Additional messages per unit per day for Standard tier	Unlimited	Unlimited

To request an update to your subscription's default limits, open a support ticket.

Azure VMware Solution limits

The following table describes the maximum limits for Azure VMware Solution.

RESOURCE	LIMIT
Clusters per private cloud	12
Minimum number of hosts per cluster	3
Maximum number of hosts per cluster	16
hosts per private cloud	96
vCenter per private cloud	1
HCX site pairings	3 with Advanced edition, 10 with Enterprise edition
AVS ExpressRoute max linked private clouds	4 The virtual network gateway used determines the actual max linked private clouds. For more details, see About ExpressRoute virtual network gateways
AVS ExpressRoute portspeed	10 Gbps The virtual network gateway used determines the actual bandwidth. For more details, see About ExpressRoute virtual network gateways
Public IPs exposed via vWAN	100
vSAN capacity limits	75% of total usable (keep 25% available for SLA)

For other VMware specific limits please use the [VMware configuration maximum tool!](#)

Backup limits

For a summary of Azure Backup support settings and limitations, see [Azure Backup Support Matrices](#).

Batch limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure Batch accounts per region per subscription	1-3	50
Dedicated cores per Batch account	90-900	Contact support
Low-priority cores per Batch account	10-100	Contact support
Active jobs and job schedules per Batch account (completed jobs have no limit)	100-300	1,000 ¹
Pools per Batch account	20-100	500 ¹

¹To request an increase beyond this limit, contact Azure Support.

NOTE

Default limits vary depending on the type of subscription you use to create a Batch account. Cores quotas shown are for Batch accounts in Batch service mode. [View the quotas in your Batch account](#).

IMPORTANT

To help us better manage capacity during the global health pandemic, the default core quotas for new Batch accounts in some regions and for some types of subscription have been reduced from the above range of values, in some cases to zero cores. When you create a new Batch account, [check your core quota](#) and [request a core quota increase](#), if required. Alternatively, consider reusing Batch accounts that already have sufficient quota.

Classic deployment model limits

If you use classic deployment model instead of the Azure Resource Manager deployment model, the following limits apply.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
vCPUs per subscription ¹	20	10,000
Coadministrators per subscription	200	200
Storage accounts per subscription ²	100	100
Cloud services per subscription	20	200
Local networks per subscription	10	500
DNS servers per subscription	9	100

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Reserved IPs per subscription	20	100
Affinity groups per subscription	256	256
Subscription name length (characters)	64	64

¹Extra small instances count as one vCPU toward the vCPU limit despite using a partial CPU core.

²The storage account limit includes both Standard and Premium storage accounts.

Container Instances limits

RESOURCE	LIMIT
Standard sku container groups per region per subscription	100 ¹
Dedicated sku container groups per region per subscription	0 ¹
Number of containers per container group	60
Number of volumes per container group	20
Standard sku cores (CPUs) per region per subscription	10 ^{1,2}
Standard sku cores (CPUs) for K80 GPU per region per subscription	18 ^{1,2}
Standard sku cores (CPUs) for P100 or V100 GPU per region per subscription	0 ^{1,2}
Ports per IP	5
Container instance log size - running instance	4 MB
Container instance log size - stopped instance	16 KB or 1,000 lines
Container group creates per hour	300 ¹
Container group creates per 5 minutes	100 ¹
Container group deletes per hour	300 ¹
Container group deletes per 5 minutes	100 ¹

¹To request a limit increase, create an [Azure Support request](#). Free subscriptions including [Azure Free Account](#) and [Azure for Students](#) aren't eligible for limit or quota increases. If you have a free subscription, you can [upgrade](#) to a Pay-As-You-Go subscription.

²Default limit for [Pay-As-You-Go](#) subscription. Limit may differ for other category types.

Container Registry limits

The following table details the features and limits of the Basic, Standard, and Premium [service tiers](#).

RESOURCE	BASIC	STANDARD	PREMIUM
Included storage ¹ (GiB)	10	100	500
Storage limit (TiB)	20	20	20
Maximum image layer size (GiB)	200	200	200
ReadOps per minute ^{2, 3}	1,000	3,000	10,000
WriteOps per minute ^{2, 4}	100	500	2,000
Download bandwidth ² (Mbps)	30	60	100
Upload bandwidth ² (Mbps)	10	20	50
Webhooks	2	10	500
Geo-replication	N/A	N/A	Supported
Availability zones	N/A	N/A	Preview
Content trust	N/A	N/A	Supported
Private link with private endpoints	N/A	N/A	Supported
• Private endpoints	N/A	N/A	10
Public IP network rules	N/A	N/A	100
Service endpoint VNet access	N/A	N/A	Preview
Customer-managed keys	N/A	N/A	Supported
Repository-scoped permissions	N/A	N/A	Preview
• Tokens	N/A	N/A	20,000
• Scope maps	N/A	N/A	20,000
• Repositories per scope map	N/A	N/A	500

¹ Storage included in the daily rate for each tier. Additional storage may be used, up to the registry storage limit, at an additional daily rate per GiB. For rate information, see [Azure Container Registry pricing](#). If you need storage beyond the registry storage limit, please contact Azure Support.

² *ReadOps*, *WriteOps*, and *Bandwidth* are minimum estimates. Azure Container Registry strives to improve

performance as usage requires.

³A [docker pull](#) translates to multiple read operations based on the number of layers in the image, plus the manifest retrieval.

⁴A [docker push](#) translates to multiple write operations, based on the number of layers that must be pushed. A `docker push` includes *ReadOps* to retrieve a manifest for an existing image.

Content Delivery Network limits

RESOURCE	LIMIT
Azure Content Delivery Network profiles	25
Content Delivery Network endpoints per profile	25
Custom domains per endpoint	25

A Content Delivery Network subscription can contain one or more Content Delivery Network profiles. A Content Delivery Network profile can contain one or more Content Delivery Network endpoints. You might want to use multiple profiles to organize your Content Delivery Network endpoints by internet domain, web application, or some other criteria.

Data Factory limits

Azure Data Factory is a multitenant service that has the following default limits in place to make sure customer subscriptions are protected from each other's workloads. To raise the limits up to the maximum for your subscription, contact support.

Version 2

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a data factory	5,000	Contact support.
Total CPU cores for Azure-SSIS Integration Runtimes under one subscription	256	Contact support.
Concurrent pipeline runs per data factory that's shared among all pipelines in the factory	10,000	10,000
Concurrent External activity runs per subscription per Azure Integration Runtime region External activities are managed on integration runtime but execute on linked services, including Databricks, stored procedure, Web, and others. This limit does not apply to Self-hosted IR.	3,000	3,000

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
<p>Concurrent Pipeline activity runs per subscription per Azure Integration Runtime region</p> <p>Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete. This limit does not apply to Self-hosted IR.</p>	1,000	1,000
<p>Concurrent authoring operations per subscription per Azure Integration Runtime region</p> <p>Including test connection, browse folder list and table list, preview data. This limit does not apply to Self-hosted IR.</p>	200	200
<p>Concurrent Data Integration Units¹ consumption per subscription per Azure Integration Runtime region</p>	<p>Region group 1²: 6,000</p> <p>Region group 2²: 3,000</p> <p>Region group 3²: 1,500</p>	<p>Region group 1²: 6,000</p> <p>Region group 2²: 3,000</p> <p>Region group 3²: 1,500</p>
<p>Maximum activities per pipeline, which includes inner activities for containers</p>	40	40
<p>Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime</p>	100	Contact support.
<p>Maximum parameters per pipeline</p>	50	50
<p>ForEach items</p>	100,000	100,000
<p>ForEach parallelism</p>	20	50
<p>Maximum queued runs per pipeline</p>	100	100
<p>Characters per expression</p>	8,192	8,192
<p>Minimum tumbling window trigger interval</p>	15 min	15 min
<p>Maximum timeout for pipeline activity runs</p>	7 days	7 days
<p>Bytes per object for pipeline objects³</p>	200 KB	200 KB
<p>Bytes per object for dataset and linked service objects³</p>	100 KB	2,000 KB
<p>Bytes per payload for each activity run⁴</p>	896 KB	896 KB
<p>Data Integration Units¹ per copy activity run</p>	256	256

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Write API calls	1,200/h	1,200/h This limit is imposed by Azure Resource Manager, not Azure Data Factory.
Read API calls	12,500/h	12,500/h This limit is imposed by Azure Resource Manager, not Azure Data Factory.
Monitoring queries per minute	1,000	1,000
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	Contact support.
Concurrent number of data flow debug sessions per user per factory	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a factory	2 GB	Contact support.

¹ The data integration unit (DIU) is used in a cloud-to-cloud copy operation, learn more from [Data integration units \(version 2\)](#). For information on billing, see [Azure Data Factory pricing](#).

² [Azure Integration Runtime](#) is [globally available](#) to ensure data compliance, efficiency, and reduced network egress costs.

REGION GROUP	REGIONS
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

³ Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

⁴ The payload for each activity run includes the activity configuration, the associated dataset(s) and linked service(s) configurations if any, and a small portion of system properties generated per activity type. Limit for this payload size doesn't relate to the amount of data you can move and process with Azure Data Factory. Learn about the [symptoms and recommendation](#) if you hit this limit.

Version 1

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Pipelines within a data factory	2,500	Contact support.
Data sets within a data factory	5,000	Contact support.
Concurrent slices per data set	10	10
Bytes per object for pipeline objects ¹	200 KB	200 KB
Bytes per object for data set and linked service objects ¹	100 KB	2,000 KB
Azure HDInsight on-demand cluster cores within a subscription ²	60	Contact support.
Cloud data movement units per copy activity run ³	32	32
Retry count for pipeline activity runs	1,000	MaxInt (32 bit)

¹ Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

² On-demand HDInsight cores are allocated out of the subscription that contains the data factory. As a result, the previous limit is the Data Factory-enforced core limit for on-demand HDInsight cores. It's different from the core limit that's associated with your Azure subscription.

³ The cloud data movement unit (DMU) for version 1 is used in a cloud-to-cloud copy operation, learn more from [Cloud data movement units \(version 1\)](#). For information on billing, see [Azure Data Factory pricing](#).

RESOURCE	DEFAULT LOWER LIMIT	MINIMUM LIMIT
Scheduling interval	15 minutes	15 minutes
Interval between retry attempts	1 second	1 second
Retry timeout value	1 second	1 second

Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource Manager API limits](#).

Data Lake Analytics limits

Azure Data Lake Analytics makes the complex task of managing distributed infrastructure and complex code easy. It dynamically provisions resources, and you can use it to do analytics on exabytes of data. When the job completes, it winds down resources automatically. You pay only for the processing power that was used. As you increase or decrease the size of data stored or the amount of compute used, you don't have to rewrite code. To raise the default limits for your subscription, contact support.

RESOURCE	LIMIT	COMMENTS
Maximum number of concurrent jobs	20	
Maximum number of analytics units (AUs) per account	250	Use any combination of up to a maximum of 250 AUs across 20 jobs. To increase this limit, contact Microsoft Support.
Maximum script size for job submission	3 MB	
Maximum number of Data Lake Analytics accounts per region per subscription	5	To increase this limit, contact Microsoft Support.

Data Lake Storage limits

Azure Data Lake Storage Gen2 is not a dedicated service or storage account type. It is the latest release of capabilities that are dedicated to big data analytics. These capabilities are available in a general-purpose v2 or BlockBlobStorage storage account, and you can obtain them by enabling the **Hierarchical namespace** feature of the account. For scale targets, see these articles.

- [Scale targets for Blob storage.](#)
- [Scale targets for standard storage accounts.](#)

Azure Data Lake Storage Gen1 is a dedicated service. It's an enterprise-wide hyper-scale repository for big data analytic workloads. You can use Data Lake Storage Gen1 to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics. There's no limit to the amount of data you can store in a Data Lake Storage Gen1 account.

RESOURCE	LIMIT	COMMENTS
Maximum number of Data Lake Storage Gen1 accounts, per subscription, per region	10	To request an increase for this limit, contact support.
Maximum number of access ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries.
Maximum number of default ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries.

Data Share limits

Azure Data Share enables organizations to simply and securely share data with their customers and partners.

RESOURCE	LIMIT
Maximum number of Data Share resources per Azure subscription	100
Maximum number of sent shares per Data Share resource	200

RESOURCE	LIMIT
Maximum number of received shares per Data Share resource	100
Maximum number of invitations per sent share	200
Maximum number of share subscriptions per sent share	200
Maximum number of datasets per share	200
Maximum number of snapshot schedules per share	1

Database Migration Service Limits

Azure Database Migration Service is a fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms with minimal downtime.

RESOURCE	LIMIT	COMMENTS
Maximum number of services per subscription, per region	10	To request an increase for this limit, contact support.

Digital Twins limits

NOTE

Some areas of this service have adjustable limits, and others do not. This is represented in the tables below with the *Adjustable?* column. When the limit can be adjusted, the *Adjustable?* value is *Yes*.

Functional limits

The following table lists the functional limits of Azure Digital Twins.

TIP

For modeling recommendations to operate within these functional limits, see [Modeling best practices](#).

AREA	CAPABILITY	DEFAULT LIMIT	ADJUSTABLE?
Azure resource	Number of Azure Digital Twins instances in a region, per subscription	10	Yes
Digital twins	Number of twins in an Azure Digital Twins instance	200,000	Yes
Digital twins	Number of incoming relationships to a single twin	5,000	No

AREA	CAPABILITY	DEFAULT LIMIT	ADJUSTABLE?
Digital twins	Number of outgoing relationships from a single twin	5,000	No
Digital twins	Maximum size (of JSON body in a PUT or PATCH request) of a single twin	32 KB	No
Digital twins	Maximum request payload size	32 KB	No
Routing	Number of endpoints for a single Azure Digital Twins instance	6	No
Routing	Number of routes for a single Azure Digital Twins instance	6	Yes
Models	Number of models within a single Azure Digital Twins instance	10,000	Yes
Models	Number of models that can be uploaded in a single API call	250	No
Models	Maximum size (of JSON body in a PUT or PATCH request) of a single model	1 MB	No
Models	Number of items returned in a single page	100	No
Query	Number of items returned in a single page	100	Yes
Query	Number of <code>AND</code> / <code>OR</code> expressions in a query	50	Yes
Query	Number of array items in an <code>IN</code> / <code>NOT IN</code> clause	50	Yes
Query	Number of characters in a query	8,000	Yes
Query	Number of <code>JOINS</code> in a query	5	Yes

Rate limits

The following table reflects the rate limits of different APIs.

API	CAPABILITY	DEFAULT LIMIT	ADJUSTABLE?
Models API	Number of requests per second	100	Yes
Digital Twins API	Number of read requests per second	1,000	Yes
Digital Twins API	Number of patch requests per second	1,000	Yes
Digital Twins API	Number of create/delete operations per second across all twins and relationships	50	Yes
Digital Twins API	Number of create/update/delete operations per second on a single twin or its relationships	10	No
Query API	Number of requests per second	500	Yes
Query API	Query Units per second	4,000	Yes
Event Routes API	Number of requests per second	100	Yes

Other limits

Limits on data types and fields within DTDL documents for Azure Digital Twins models can be found within its spec documentation in GitHub: [Digital Twins Definition Language \(DTDLE\) - version 2](#).

Query latency details are described in [Concepts: Query language](#). Limitations of particular query language features can be found in the [query reference documentation](#).

Event Grid limits

The following limits apply to Azure Event Grid **topics** (system, custom, and partner topics).

RESOURCE	LIMIT
Custom topics per Azure subscription	100
Event subscriptions per topic	500
Publish rate for a custom or a partner topic (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)
Event size	1 MB
Private endpoint connections per topic	64
IP Firewall rules per topic	16

The following limits apply to Azure Event Grid **domains**.

RESOURCE	LIMIT
Topics per event domain	100,000
Event subscriptions per topic within a domain	500
Domain scope event subscriptions	50
Publish rate for an event domain (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)
Event Domains per Azure Subscription	100
Private endpoint connections per domain	64
IP Firewall rules per domain	16

Event Hubs limits

The following tables provide quotas and limits specific to [Azure Event Hubs](#). For information about Event Hubs pricing, see [Event Hubs pricing](#).

Common limits for all tiers

The following limits are common across all tiers.

LIMIT	NOTES	VALUE
Size of an event hub name	-	256 characters
Size of a consumer group name	Kafka protocol doesn't require the creation of a consumer group.	Kafka: 256 characters AMQP: 50 characters
Number of non-epoch receivers per consumer group	-	5
Number of authorization rules per namespace	Subsequent requests for authorization rule creation are rejected.	12
Number of calls to the GetRuntimeInformation method	-	50 per second
Number of virtual networks (VNet)	-	128
Number of IP Config rules	-	128
Maximum length of a schema group name		50
Maximum length of a schema name		100
Size in bytes per schema		1 MB

LIMIT	NOTES	VALUE
Number of properties per schema group		1024
Size in bytes per schema group property key		256
Size in bytes per schema group property value		1024

Basic vs. standard vs. premium vs. dedicated tiers

The following table shows limits that may be different for basic, standard, and dedicated tiers. In the table CU is [capacity unit](#), PU is [processing unit](#), TU is [throughput unit](#).

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Maximum size of Event Hubs publication	256 KB	1 MB	1 MB	1 MB
Number of consumer groups per event hub	1	20	100	1000 No limit per CU
Number of brokered connections per namespace	100	5,000	10000 per processing unit per PU	100, 000 per CU
Maximum retention period of event data	1 day	7 days	90 days 1 TB per PU	90 days 10 TB per CU
Maximum TUs or PUs or CUs	20 TUs	40 TUs	16 PUs	20 CUs
Number of partitions per event hub	32	32	100 per event hub 200 per PU	1024 per event hub 2000 per CU
Number of namespaces per subscription	1000	1000	1000	1000 (50 per CU)
Number of event hubs per namespace	10	10	100	1000
Capture	N/A	Pay per hour	Included	Included
Size of the schema registry (namespace) in mega bytes	N/A	25	100	1024
Number of schema groups in a schema registry or namespace	N/A	1 - excluding the default group	100 1 MB per schema	1000 1 MB per schema

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Number of schema versions across all schema groups	N/A	25	1000	10000
Throughput per unit	Ingress - 1 MB/s or 1000 events per second Egress - 2 Mb/s or 4096 events per second	Ingress - 1 MB/s or 1000 events per second Egress - 2 Mb/s or 4096 events per second	No limits per PU *	No limits per CU *

* Depends on various factors such as resource allocation, number of partitions, storage and so on.

NOTE

You can publish events individually or batched. The publication limit (according to SKU) applies regardless of whether it is a single event or a batch. Publishing events larger than the maximum threshold will be rejected.

IoT Central limits

IoT Central limits the number of applications you can deploy in a subscription to 10. If you need to increase this limit, contact [Microsoft support](#).

IoT Hub limits

The following table lists the limits associated with the different service tiers S1, S2, S3, and F1. For information about the cost of each *unit* in each tier, see [Azure IoT Hub pricing](#).

RESOURCE	S1 STANDARD	S2 STANDARD	S3 STANDARD	F1 FREE
Messages/day	400,000	6,000,000	300,000,000	8,000
Maximum units	200	200	10	1

NOTE

If you anticipate using more than 200 units with an S1 or S2 tier hub or 10 units with an S3 tier hub, contact Microsoft Support.

The following table lists the limits that apply to IoT Hub resources.

RESOURCE	LIMIT
Maximum paid IoT hubs per Azure subscription	50
Maximum free IoT hubs per Azure subscription	1
Maximum number of characters in a device ID	128
Maximum number of device identities returned in a single call	1,000

RESOURCE	LIMIT
IoT Hub message maximum retention for device-to-cloud messages	7 days
Maximum size of device-to-cloud message	256 KB
Maximum size of device-to-cloud batch	AMQP and HTTP: 256 KB for the entire batch MQTT: 256 KB for each message
Maximum messages in device-to-cloud batch	500
Maximum size of cloud-to-device message	64 KB
Maximum TTL for cloud-to-device messages	2 days
Maximum delivery count for cloud-to-device messages	100
Maximum cloud-to-device queue depth per device	50
Maximum delivery count for feedback messages in response to a cloud-to-device message	100
Maximum TTL for feedback messages in response to a cloud-to-device message	2 days
Maximum size of device twin	8 KB for tags section, and 32 KB for desired and reported properties sections each
Maximum length of device twin string key	1 KB
Maximum length of device twin string value	4 KB
Maximum depth of object in device twin	10
Maximum size of direct method payload	128 KB
Job history maximum retention	30 days
Maximum concurrent jobs	10 (for S3), 5 for (S2), 1 (for S1)
Maximum additional endpoints	10 (for S1, S2, and S3)
Maximum message routing rules	100 (for S1, S2, and S3)
Maximum number of concurrently connected device streams	50 (for S1, S2, S3, and F1 only)
Maximum device stream data transfer	300 MB per day (for S1, S2, S3, and F1 only)

NOTE

If you need more than 50 paid IoT hubs in an Azure subscription, contact Microsoft Support.

NOTE

Currently, the total number of devices plus modules that can be registered to a single IoT hub is capped at 1,000,000. If you want to increase this limit, contact [Microsoft Support](#).

IoT Hub throttles requests when the following quotas are exceeded.

THROTTLE	PER-HUB VALUE
Identity registry operations (create, retrieve, list, update, and delete), individual or bulk import/export	83.33/sec/unit (5,000/min/unit) (for S3). 1.67/sec/unit (100/min/unit) (for S1 and S2).
Device connections	6,000/sec/unit (for S3), 120/sec/unit (for S2), 12/sec/unit (for S1). Minimum of 100/sec.
Device-to-cloud sends	6,000/sec/unit (for S3), 120/sec/unit (for S2), 12/sec/unit (for S1). Minimum of 100/sec.
Cloud-to-device sends	83.33/sec/unit (5,000/min/unit) (for S3), 1.67/sec/unit (100/min/unit) (for S1 and S2).
Cloud-to-device receives	833.33/sec/unit (50,000/min/unit) (for S3), 16.67/sec/unit (1,000/min/unit) (for S1 and S2).
File upload operations	83.33 file upload initiations/sec/unit (5,000/min/unit) (for S3), 1.67 file upload initiations/sec/unit (100/min/unit) (for S1 and S2). 10,000 SAS URIs can be out for an Azure Storage account at one time. 10 SAS URIs/device can be out at one time.
Direct methods	24 MB/sec/unit (for S3), 480 KB/sec/unit (for S2), 160 KB/sec/unit (for S1). Based on 8-KB throttling meter size.
Device twin reads	500/sec/unit (for S3), Maximum of 100/sec or 10/sec/unit (for S2), 100/sec (for S1)
Device twin updates	250/sec/unit (for S3), Maximum of 50/sec or 5/sec/unit (for S2), 50/sec (for S1)
Jobs operations (create, update, list, and delete)	83.33/sec/unit (5,000/min/unit) (for S3), 1.67/sec/unit (100/min/unit) (for S2), 1.67/sec/unit (100/min/unit) (for S1).
Jobs per-device operation throughput	50/sec/unit (for S3), maximum of 10/sec or 1/sec/unit (for S2), 10/sec (for S1).
Device stream initiation rate	5 new streams/sec (for S1, S2, S3, and F1 only).

IoT Hub Device Provisioning Service limits

NOTE

Some areas of this service have adjustable limits. This is represented in the tables below with the *Adjustable?* column. When the limit can be adjusted, the *Adjustable?* value is *Yes*.

If your business requires raising an adjustable limit or quota above the default limit, you can request additional resources by [opening a support ticket](#).

The following table lists the limits that apply to Azure IoT Hub Device Provisioning Service resources.

RESOURCE	LIMIT	ADJUSTABLE?
Maximum device provisioning services per Azure subscription	10	Yes
Maximum number of registrations	1,000,000	Yes
Maximum number of individual enrollments	1,000,000	Yes
Maximum number of enrollment groups (<i>X.509 certificate</i>)	100	Yes
Maximum number of enrollment groups (<i>symmetric key</i>)	100	No
Maximum number of CAs	25	No
Maximum number of linked IoT hubs	50	No
Maximum size of message	96 KB	No

TIP

If the hard limit on symmetric key enrollment groups is a blocking issue, it is recommended to use individual enrollments as a workaround.

The Device Provisioning Service has the following rate limits.

RATE	PER-UNIT VALUE	ADJUSTABLE?
Operations	200/min/service	Yes
Device registrations	200/min/service	Yes
Device polling operation	5/10 sec/device	No

Each API call on DPS is billable as one *Operation*. This includes all the service APIs and the device registration API. The device registration polling operation is not billed.

Key Vault limits

Azure Key Vault service supports two resource types: Vaults and Managed HSMs. The following two sections

describe the service limits for each of them respectively.

Resource type: vault

This section describes service limits for resource type `vaults`.

Key transactions (maximum transactions allowed in 10 seconds, per vault per region¹):

KEY TYPE	HSM KEY CREATE KEY	HSM KEY ALL OTHER TRANSACTIONS	SOFTWARE KEY CREATE KEY	SOFTWARE KEY ALL OTHER TRANSACTIONS
RSA 2,048-bit	5	1,000	10	2,000
RSA 3,072-bit	5	250	10	500
RSA 4,096-bit	5	125	10	250
ECC P-256	5	1,000	10	2,000
ECC P-384	5	1,000	10	2,000
ECC P-521	5	1,000	10	2,000
ECC SECP256K1	5	1,000	10	2,000

NOTE

In the previous table, we see that for RSA 2,048-bit software keys, 2,000 GET transactions per 10 seconds are allowed. For RSA 2,048-bit HSM-keys, 1,000 GET transactions per 10 seconds are allowed.

The throttling thresholds are weighted, and enforcement is on their sum. For example, as shown in the previous table, when you perform GET operations on RSA HSM-keys, it's eight times more expensive to use 4,096-bit keys compared to 2,048-bit keys. That's because $1,000/125 = 8$.

In a given 10-second interval, an Azure Key Vault client can do *only one* of the following operations before it encounters a `429` throttling HTTP status code:

- 2,000 RSA 2,048-bit software-key GET transactions
- 1,000 RSA 2,048-bit HSM-key GET transactions
- 125 RSA 4,096-bit HSM-key GET transactions
- 124 RSA 4,096-bit HSM-key GET transactions and 8 RSA 2,048-bit HSM-key GET transactions

Secrets, managed storage account keys, and vault transactions:

TRANSACTIONS TYPE	MAXIMUM TRANSACTIONS ALLOWED IN 10 SECONDS, PER VAULT PER REGION ¹
All transactions	2,000

For information on how to handle throttling when these limits are exceeded, see [Azure Key Vault throttling guidance](#).

¹ A subscription-wide limit for all transaction types is five times per key vault limit. For example, HSM-other transactions per subscription are limited to 5,000 transactions in 10 seconds per subscription.

Backup keys, secrets, certificates

When you back up a key vault object, such as a secret, key, or certificate, the backup operation will download the

object as an encrypted blob. This blob can't be decrypted outside of Azure. To get usable data from this blob, you must restore the blob into a key vault within the same Azure subscription and Azure geography

TRANSACTIONS TYPE	MAXIMUM KEY VAULT OBJECT VERSIONS ALLOWED
Backup individual key, secret, certificate	500

NOTE

Attempting to backup a key, secret, or certificate object with more versions than above limit will result in an error. It is not possible to delete previous versions of a key, secret, or certificate.

Limits on count of keys, secrets and certificates:

Key Vault does not restrict the number of keys, secrets or certificates that can be stored in a vault. The transaction limits on the vault should be taken into account to ensure that operations are not throttled.

Key Vault does not restrict the number of versions on a secret, key or certificate, but storing a large number of versions (500+) can impact the performance of backup operations. See [Azure Key Vault Backup](#).

Azure Private Link integration

NOTE

The number of key vaults with private endpoints enabled per subscription is an adjustable limit. The limit shown below is the default limit. If you would like to request a limit increase for your service, please create a support request and it will be assessed on a case by case basis.

RESOURCE	LIMIT
Private endpoints per key vault	64
Key vaults with private endpoints per subscription	400

Resource type: Managed HSM

This section describes service limits for resource type `managed HSM`.

Object limits

ITEM	LIMITS
Number of HSM instances per subscription per region	1
Number of keys per HSM Pool	5000
Number of versions per key	100
Number of custom role definitions per HSM	50
Number of role assignments at HSM scope	50
Number of role assignment at each individual key scope	10

Transaction limits for administrative operations (number of operations per second per HSM instance)

OPERATION	NUMBER OF OPERATIONS PER SECOND
All RBAC operations (includes all CRUD operations for role definitions and role assignments)	5
Full HSM Backup/Restore (only one concurrent backup or restore operation per HSM instance supported)	1

Transaction limits for cryptographic operations (number of operations per second per HSM instance)

- Each Managed HSM instance constitutes 3 load balanced HSM partitions. The throughput limits are a function of underlying hardware capacity allocated for each partition. The tables below show maximum throughput with at least one partition available. Actual throughput may be up to 3x higher if all 3 partitions are available.
- Throughput limits noted assume that one single key is being used to achieve maximum throughput. For example, if a single RSA-2048 key is used the maximum throughput will be 1100 sign operations. If you use 1100 different keys with 1 transaction per second each, they will not be able to achieve the same throughput.

RSA key operations (number of operations per second per HSM instance)

OPERATION	2048-BIT	3072-BIT	4096-BIT
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	10000	10000	6000
Decrypt	1100	360	160
Wrap	10000	10000	6000
Unwrap	1100	360	160
Sign	1100	360	160
Verify	10000	10000	6000

EC key operations (number of operations per second per HSM instance)

This table describes number of operations per second for each curve type.

OPERATION	P-256	P-256K	P-384	P-521
Create Key	1	1	1	1

OPERATION	P-256	P-256K	P-384	P-521
Delete Key (soft-delete)	10	10	10	10
Purge Key	10	10	10	10
Backup Key	10	10	10	10
Restore Key	10	10	10	10
Get Key Information	1100	1100	1100	1100
Sign	260	260	165	56
Verify	130	130	82	28

AES key operations (number of operations per second per HSM instance)

- Encrypt and Decrypt operations assume a 4KB packet size.
- Throughput limits for Encrypt/Decrypt apply to AES-CBC and AES-GCM algorithms.
- Throughput limits for Wrap/Unwrap apply to AES-KW algorithm.

OPERATION	128-BIT	192-BIT	256-BIT
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	8000	8000	8000
Decrypt	8000	8000	8000
Wrap	9000	9000	9000
Unwrap	9000	9000	9000

Managed identity limits

- Each managed identity counts towards the object quota limit in an Azure AD tenant as described in [Azure AD service limits and restrictions](#).
- The rate at which managed identities can be created have the following limits:
 1. Per Azure AD Tenant per Azure region: 200 create operations per 20 seconds.

2. Per Azure Subscription per Azure region : 40 create operations per 20 seconds.

- When you create user-assigned managed identities, only alphanumeric characters (0-9, a-z, and A-Z) and the hyphen (-) are supported. For the assignment to a virtual machine or virtual machine scale set to work properly, the name is limited to 24 characters.

Media Services limits

NOTE

For resources that aren't fixed, open a support ticket to ask for an increase in the quotas. Don't create additional Azure Media Services accounts in an attempt to obtain higher limits.

Account limits

RESOURCE	DEFAULT LIMIT
Media Services accounts in a single subscription	100 (fixed)

Asset limits

RESOURCE	DEFAULT LIMIT
Assets per Media Services account	1,000,000

Storage (media) limits

RESOURCE	DEFAULT LIMIT
File size	In some scenarios, there is a limit on the maximum file size supported for processing in Media Services. ⁽¹⁾
Storage accounts	100 ⁽²⁾ (fixed)

¹ The maximum size supported for a single blob is currently up to 5 TB in Azure Blob Storage. Additional limits apply in Media Services based on the VM sizes that are used by the service. The size limit applies to the files that you upload and also the files that get generated as a result of Media Services processing (encoding or analyzing). If your source file is larger than 260-GB, your Job will likely fail.

The following table shows the limits on the media reserved units S1, S2, and S3. If your source file is larger than the limits defined in the table, your encoding job fails. If you encode 4K resolution sources of long duration, you're required to use S3 media reserved units to achieve the performance needed. If you have 4K content that's larger than the 260-GB limit on the S3 media reserved units, open a support ticket.

MEDIA RESERVED UNIT TYPE	MAXIMUM INPUT SIZE (GB)
S1	26
S2	60
S3	260

² The storage accounts must be from the same Azure subscription.

Jobs (encoding & analyzing) limits

RESOURCE	DEFAULT LIMIT
Jobs per Media Services account	500,000 ⁽³⁾ (fixed)
Job inputs per Job	50 (fixed)
Job outputs per Job	20 (fixed)
Transforms per Media Services account	100 (fixed)
Transform outputs in a Transform	20 (fixed)
Files per job input	10 (fixed)

³ This number includes queued, finished, active, and canceled Jobs. It does not include deleted Jobs.

Any Job record in your account older than 90 days will be automatically deleted, even if the total number of records is below the maximum quota.

Live streaming limits

RESOURCE	DEFAULT LIMIT
Live Events ⁽⁴⁾ per Media Services account	5
Live Outputs per Live Event	3 ⁽⁵⁾
Max Live Output duration	Size of the DVR window

⁴ For detailed information about Live Event limitations, see [Live Event types comparison and limitations](#).

⁵ Live Outputs start on creation and stop when deleted.

Packaging & delivery limits

RESOURCE	DEFAULT LIMIT
Streaming Endpoints (stopped or running) per Media Services account	2
Dynamic Manifest Filters	100
Streaming Policies	100 ⁽⁶⁾
Unique Streaming Locators associated with an Asset at one time	100 ⁽⁷⁾ (fixed)

⁶ When using a custom [Streaming Policy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. You should not be creating a new Streaming Policy for each Streaming Locator.

⁷ Streaming Locators are not designed for managing per-user access control. To give different access rights to individual users, use Digital Rights Management (DRM) solutions.

Protection limits

RESOURCE	DEFAULT LIMIT
Options per Content Key Policy	30
Licenses per month for each of the DRM types on Media Services key delivery service per account	1,000,000

Support ticket

For resources that are not fixed, you may ask for the quotas to be raised, by opening a [support ticket](#). Include detailed information in the request on the desired quota changes, use-case scenarios, and regions required. Do **not** create additional Azure Media Services accounts in an attempt to obtain higher limits.

Media Services v2 (legacy)

For limits specific to Media Services v2 (legacy), see [Media Services v2 \(legacy\)](#)

Mobile Services limits

TIER	FREE	BASIC	STANDARD
API calls	500,000	1.5 million per unit	15 million per unit
Active devices	500	Unlimited	Unlimited
Scale	N/A	Up to 6 units	Unlimited units
Push notifications	Azure Notification Hubs Free tier included, up to 1 million pushes	Notification Hubs Basic tier included, up to 10 million pushes	Notification Hubs Standard tier included, up to 10 million pushes
Real-time messaging/ Web Sockets	Limited	350 per mobile service	Unlimited
Offline synchronizations	Limited	Included	Included
Scheduled jobs	Limited	Included	Included
Azure SQL Database (required) Standard rates apply for additional capacity	20 MB included	20 MB included	20 MB included
CPU capacity	60 minutes per day	Unlimited	Unlimited
Outbound data transfer	165 MB per day (daily rollover)	Included	Included

For more information on limits and pricing, see [Azure Mobile Services pricing](#).

Multi-Factor Authentication limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum number of trusted IP addresses or ranges per subscription	0	50
Remember my devices, number of days	14	60
Maximum number of app passwords	0	No limit
Allow X attempts during MFA call	1	99
Two-way text message timeout seconds	60	600
Default one-time bypass seconds	300	1,800
Lock user account after X consecutive MFA denials	Not set	99
Reset account lockout counter after X minutes	Not set	9,999
Unlock account after X minutes	Not set	9,999

Networking limits

Networking limits - Azure Resource Manager

The following limits apply only for networking resources managed through **Azure Resource Manager** per region per subscription. Learn how to [view your current resource usage against your subscription limits](#).

NOTE

We recently increased all default limits to their maximum limits. If there's no maximum limit column, the resource doesn't have adjustable limits. If you had these limits increased by support in the past and don't see updated limits in the following tables, [open an online customer support request at no charge](#)

RESOURCE	LIMIT
Virtual networks	1,000
Subnets per virtual network	3,000
Virtual network peerings per virtual network	500
Virtual network gateways (VPN gateways) per virtual network	1
Virtual network gateways (ExpressRoute gateways) per virtual network	1
DNS servers per virtual network	20

RESOURCE	LIMIT
Private IP addresses per virtual network	65,536
Private IP addresses per network interface	256
Private IP addresses per virtual machine	256
Public IP addresses per network interface	256
Public IP addresses per virtual machine	256
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500,000
Network interface cards	65,536
Network Security Groups	5,000
NSG rules per NSG	1,000
IP addresses and ranges specified for source or destination in a security group	4,000
Application security groups	3,000
Application security groups per IP configuration, per NIC	20
IP configurations per application security group	4,000
Application security groups that can be specified within all security rules of a network security group	100
User-defined route tables	200
User-defined routes per route table	400
Point-to-site root certificates per Azure VPN Gateway	20
Point-to-site revoked client certificates per Azure VPN Gateway	300
Virtual network TAPs	100
Network interface TAP configurations per virtual network TAP	100

Public IP address limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Public IP addresses ^{1,2}	10 for Basic.	Contact support.
Static Public IP addresses ¹	10 for Basic.	Contact support.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Standard Public IP addresses ¹	10	Contact support.
Public IP addresses per Resource Group	800	Contact support.
Public IP Prefixes	limited by number of Standard Public IPs in a subscription	Contact support.
Public IP prefix length	/28	Contact support.

¹Default limits for Public IP addresses vary by offer category type, such as Free Trial, Pay-As-You-Go, CSP. For example, the default for Enterprise Agreement subscriptions is 1000.

²Public IP addresses limit refers to the total amount of Public IP addresses, including Basic and Standard.

Load balancer limits

The following limits apply only for networking resources managed through Azure Resource Manager per region per subscription. Learn how to [view your current resource usage against your subscription limits](#).

Standard Load Balancer

RESOURCE	LIMIT
Load balancers	1,000
Rules (Load Balancer + Inbound NAT) per resource	1,500
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations	600
Backend pool size	1,000 IP configurations, single virtual network
Backend resources per Load Balancer ¹	1,200
High-availability ports	1 per internal frontend
Outbound rules per Load Balancer	600
Load Balancers per VM	2 (1 Public and 1 internal)

¹ The limit is up to 1,200 resources, in any combination of standalone virtual machine resources, availability set resources, and virtual machine scale-set placement groups.

Basic Load Balancer

RESOURCE	LIMIT
Load balancers	1,000
Rules per resource	250

RESOURCE	LIMIT
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations ²	200
Backend pool size	300 IP configurations, single availability set
Availability sets per Load Balancer	1
Load Balancers per VM	2 (1 Public and 1 internal)

² The limit for a single discrete resource in a backend pool (standalone virtual machine, availability set, or virtual machine scale-set placement group) is to have up to 250 Frontend IP configurations across a single Basic Public Load Balancer and Basic Internal Load Balancer.

The following limits apply only for networking resources managed through the classic deployment model per subscription. Learn how to [view your current resource usage against your subscription limits](#).

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Virtual networks	100	100
Local network sites	20	50
DNS servers per virtual network	20	20
Private IP addresses per virtual network	4,096	4,096
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500,000, up to 1,000,000 for two or more NICs.	500,000, up to 1,000,000 for two or more NICs.
Network Security Groups (NSGs)	200	200
NSG rules per NSG	200	1,000
User-defined route tables	200	200
User-defined routes per route table	400	400
Public IP addresses (dynamic)	500	500
Reserved public IP addresses	500	500
Public IP per deployment	5	Contact support
Private IP (internal load balancing) per deployment	1	1
Endpoint access control lists (ACLs)	50	50

ExpressRoute limits

RESOURCE	LIMIT
ExpressRoute circuits per subscription	10
ExpressRoute circuits per region per subscription, with Azure Resource Manager	10
Maximum number of routes advertised to Azure private peering with ExpressRoute Standard	4,000
Maximum number of routes advertised to Azure private peering with ExpressRoute Premium add-on	10,000
Maximum number of routes advertised from Azure private peering from the VNet address space for an ExpressRoute connection	1,000
Maximum number of routes advertised to Microsoft peering with ExpressRoute Standard	200
Maximum number of routes advertised to Microsoft peering with ExpressRoute Premium add-on	200
Maximum number of ExpressRoute circuits linked to the same virtual network in the same peering location	4
Maximum number of ExpressRoute circuits linked to the same virtual network in different peering locations	16 (For more information, see Gateway SKU .)
Number of virtual network links allowed per ExpressRoute circuit	See the Number of virtual networks per ExpressRoute circuit table.

Number of virtual networks per ExpressRoute circuit

CIRCUIT SIZE	NUMBER OF VIRTUAL NETWORK LINKS FOR STANDARD	NUMBER OF VIRTUAL NETWORK LINKS WITH PREMIUM ADD-ON
50 Mbps	10	20
100 Mbps	10	25
200 Mbps	10	25
500 Mbps	10	40
1 Gbps	10	50
2 Gbps	10	60
5 Gbps	10	75
10 Gbps	10	100
40 Gbps*	10	100

CIRCUIT SIZE	NUMBER OF VIRTUAL NETWORK LINKS FOR STANDARD	NUMBER OF VIRTUAL NETWORK LINKS WITH PREMIUM ADD-ON
100 Gbps*	10	100

* 100 Gbps ExpressRoute Direct Only

NOTE

Global Reach connections count against the limit of virtual network connections per ExpressRoute Circuit. For example, a 10 Gbps Premium Circuit would allow for 5 Global Reach connections and 95 connections to the ExpressRoute Gateways or 95 Global Reach connections and 5 connections to the ExpressRoute Gateways or any other combination up to the limit of 100 connections for the circuit.

Virtual Network Gateway limits

RESOURCE	LIMIT
VNet Address Prefixes	600 per VPN gateway
Aggregate BGP routes	4,000 per VPN gateway
Local Network Gateway address prefixes	1000 per local network gateway
S2S connections	Depends on the gateway SKU
P2S connections	Depends on the gateway SKU
P2S route limit - IKEv2	256 for non-Windows / 25 for Windows
P2S route limit - OpenVPN	1000
Max. flows	100K for VpnGw1/AZ / 512K for VpnGw2-4/AZ

NAT Gateway limits

RESOURCE	LIMIT
Public IP addresses	16 per NAT gateway

Virtual WAN limits

RESOURCE	LIMIT
Virtual WAN hubs per virtual wan	Azure regions
VPN (branch) connections per hub	1,000
Aggregate throughput per Virtual WAN Site-to-site VPN gateway	20 Gbps
Throughput per Virtual WAN VPN connection (2 tunnels)	2 Gbps with 1 Gbps/IPsec tunnel

RESOURCE	LIMIT
Point-to-Site users per hub	100,000
Aggregate throughput per Virtual WAN User VPN (Point-to-site) gateway	200 Gbps
Aggregate throughput per Virtual WAN ExpressRoute gateway	20 Gbps
ExpressRoute Circuit connections per hub	8
VNet connections per hub	500 minus total number of hubs in Virtual WAN
Aggregate throughput per Virtual WAN Hub Router	50 Gbps for VNet to VNet transit
VM workload across all VNets connected to a single Virtual WAN hub	2000 (If you want to raise the limit or quota above the default limit, open an online customer support request.)

Application Gateway limits

The following table applies to v1, v2, Standard, and WAF SKUs unless otherwise stated.

RESOURCE	LIMIT	NOTE
Azure Application Gateway	1,000 per subscription	
Front-end IP configurations	2	1 public and 1 private
Front-end ports	100 ¹	
Back-end address pools	100 ¹	
Back-end servers per pool	1,200	
HTTP listeners	200 ¹	Limited to 100 active listeners that are routing traffic. Active listeners = total number of listeners - listeners not active. If a default configuration inside a routing rule is set to route traffic (for example, it has a listener, a backend pool, and HTTP settings) then that also counts as a listener.
HTTP load-balancing rules	100 ¹	
Back-end HTTP settings	100 ¹	
Instances per gateway	V1 SKU - 32 V2 SKU - 125	
SSL certificates	100 ¹	1 per HTTP listener

RESOURCE	LIMIT	NOTE
Maximum SSL certificate size	V1 SKU - 10 KB V2 SKU - 16 KB	
Authentication certificates	100	
Trusted root certificates	100	
Request timeout minimum	1 second	
Request timeout maximum	24 hours	
Number of sites	100 ¹	1 per HTTP listener
URL maps per listener	1	
Maximum path-based rules per URL map	100	
Redirect configurations	100 ¹	
Number of rewrite rule sets	400	
Number of Header or URL configuration per rewrite rule set	40	
Number of conditions per rewrite rule set	40	
Concurrent WebSocket connections	Medium gateways 20k ² Large gateways 50k ²	
Maximum URL length	32KB	
Maximum header size for HTTP/2	16KB	
Maximum file upload size, Standard	2 GB	
Maximum file upload size WAF	V1 Medium WAF gateways, 100 MB V1 Large WAF gateways, 500 MB V2 WAF, 750 MB	
WAF body size limit, without files	128 KB	
Maximum WAF custom rules	100	
Maximum WAF exclusions per Application Gateway	40	

¹ In case of WAF-enabled SKUs, you must limit the number of resources to 40.

² Limit is per Application Gateway instance not per Application Gateway resource.

Network Watcher limits

RESOURCE	LIMIT	NOTE
Azure Network Watcher	1 per region	Network Watcher is created to enable access to the service. Only one instance of Network Watcher is required per subscription per region.
Packet capture sessions	10,000 per region	Number of sessions only, not saved captures.

Private Link limits

The following limits apply to Azure private link:

RESOURCE	LIMIT
Number of private endpoints per virtual network	1000
Number of private endpoints per subscription	64000
Number of private link services per subscription	800
Number of IP Configurations on a private link service	8 (This number is for the NAT IP addresses used per PLS)
Number of private endpoints on the same private link service	1000
Number of private endpoints per key vault	64
Number of key vaults with private endpoints per subscription	400
Number of private DNS zone groups that can be linked to a private endpoint	1
Number of DNS zones in each group	5

Purview limits

The latest values for Azure Purview quotas can be found in the [Azure Purview quota page](#)

Traffic Manager limits

RESOURCE	LIMIT
Profiles per subscription	200
Endpoints per profile	200

Azure Bastion limits

WORKLOAD TYPE*	LIMIT**
Light	100

WORKLOAD TYPE*	LIMIT**
Medium	50
Heavy	5

*These workload types are defined here: [Remote Desktop workloads](#)

**These limits are based on RDP performance tests for Azure Bastion. The numbers may vary due to other on-going RDP sessions or other on-going SSH sessions.

Azure DNS limits

Public DNS zones

RESOURCE	LIMIT
Public DNS Zones per subscription	250 ¹
Record sets per public DNS zone	10,000 ¹
Records per record set in public DNS zone	20
Number of Alias records for a single Azure resource	20

¹If you need to increase these limits, contact Azure Support.

Private DNS zones

RESOURCE	LIMIT
Private DNS zones per subscription	1000
Record sets per private DNS zone	25000
Records per record set for private DNS zones	20
Virtual Network Links per private DNS zone	1000
Virtual Networks Links per private DNS zones with auto-registration enabled	100
Number of private DNS zones a virtual network can get linked to with auto-registration enabled	1
Number of private DNS zones a virtual network can get linked	1000
Number of DNS queries a virtual machine can send to Azure DNS resolver, per second	1000 ¹
Maximum number of DNS queries queued (pending response) per virtual machine	200 ¹

¹These limits are applied to every individual virtual machine and not at the virtual network level. DNS queries exceeding these limits are dropped.

Azure Firewall limits

RESOURCE	LIMIT
Data throughput	30 Gbps
Rule limits	10,000 unique source/destinations in network rules
Maximum DNAT rules	298 for a single public IP address. Any additional public IP addresses contribute to the available SNAT ports, but reduce the number of the available DNAT rules. For example, two public IP addresses allow for 297 DNAT rules. If a rule's protocol is configured for both TCP and UDP, it counts as two rules.
Minimum AzureFirewallSubnet size	/26
Port range in network and application rules	1 - 65535
Public IP addresses	250 maximum. All public IP addresses can be used in DNAT rules and they all contribute to available SNAT ports.
IP addresses in IP Groups	Maximum of 100 IP Groups per firewall. Maximum 5000 individual IP addresses or IP prefixes per each IP Group.
Route table	<p>By default, AzureFirewallSubnet has a 0.0.0.0/0 route with the NextHopType value set to Internet.</p> <p>Azure Firewall must have direct Internet connectivity. If your AzureFirewallSubnet learns a default route to your on-premises network via BGP, you must override that with a 0.0.0.0/0 UDR with the NextHopType value set as Internet to maintain direct Internet connectivity. By default, Azure Firewall doesn't support forced tunneling to an on-premises network.</p> <p>However, if your configuration requires forced tunneling to an on-premises network, Microsoft will support it on a case by case basis. Contact Support so that we can review your case. If accepted, we'll allow your subscription and ensure the required firewall Internet connectivity is maintained.</p>
FQDNs in network rules	For good performance, do not exceed more than 1000 FQDNs across all network rules per firewall.

Azure Front Door Service limits

RESOURCE	LIMIT
Azure Front Door resources per subscription	100
Front-end hosts, which includes custom domains per resource	500
Routing rules per resource	500
Back-end pools per resource	50

RESOURCE	LIMIT
Back ends per back-end pool	100
Path patterns to match for a routing rule	25
URLs in a single cache purge call	100
Custom web application firewall rules per policy	100
Web application firewall policy per subscription	100
Web application firewall match conditions per custom rule	10
Web application firewall IP address ranges per match condition	600
Web application firewall string match values per match condition	10
Web application firewall string match value length	256
Web application firewall POST body parameter name length	256
Web application firewall HTTP header name length	256
Web application firewall cookie name length	256
Web application firewall HTTP request body size inspected	128 KB
Web application firewall custom response body length	2 KB

Azure Front Door Standard/Premium (Preview) Service Limits

*** Maximum 500 total Standard and Premium profiles per subscription.

RESOURCE	STANDARD SKU LIMIT	PREMIUM SKU LIMIT
Maximum endpoint per profile	10	25
Maximum custom domain per profile	100	200
Maximum origin group per profile	100	200
Maximum secrets per profile	100	200
Maximum security policy per profile	100	200
Maximum rule set per profile	100	200
Maximum rules per rule set	100	100
Maximum origin per origin group	50	50

RESOURCE	STANDARD SKU LIMIT	PREMIUM SKU LIMIT
Maximum routes per endpoint	100	200
URLs in a single cache purge call	100	100
Custom web application firewall rules per policy	100	100
Web application firewall match conditions per custom rule	10	10
Web application firewall IP address ranges per match condition	600	600
Web application firewall string match values per match condition	10	10
Web application firewall string match value length	256	256
Web application firewall POST body parameter name length	256	256
Web application firewall HTTP header name length	256	256
Web application firewall cookie name length	256	256
Web application firewall HTTP request body size inspected	128 KB	128 KB
Web application firewall custom response body length	2 KB	2 KB

Timeout values

Client to Front Door

- Front Door has an idle TCP connection timeout of 61 seconds.

Front Door to application back-end

- If the response is a chunked response, a 200 is returned if or when the first chunk is received.
- After the HTTP request is forwarded to the back end, Front Door waits for 30 seconds for the first packet from the back end. Then it returns a 503 error to the client. This value is configurable via the field `sendRecvTimeoutSeconds` in the API.
 - For caching scenarios, this timeout is not configurable and so, if a request is cached and it takes more than 30 seconds for the first packet from Front Door or from the backend, then a 504 error is returned to the client.
- After the first packet is received from the back end, Front Door waits for 30 seconds in an idle timeout. Then it returns a 503 error to the client. This timeout value is not configurable.
- Front Door to the back-end TCP session timeout is 90 seconds.

Upload and download data limit

	WITH CHUNKED TRANSFER ENCODING (CTE)	WITHOUT HTTP CHUNKING
Download	There's no limit on the download size.	There's no limit on the download size.
Upload	There's no limit as long as each CTE upload is less than 2 GB.	The size can't be larger than 2 GB.

Other limits

- Maximum URL size - 8,192 bytes - Specifies maximum length of the raw URL (scheme + hostname + port + path + query string of the URL)
- Maximum Query String size - 4,096 bytes - Specifies the maximum length of the query string, in bytes.
- Maximum HTTP response header size from health probe URL - 4,096 bytes - Specified the maximum length of all the response headers of health probes.
- Maximum rules engine action header value character: 640 characters.
- Maximum rules engine condition header value character: 256 characters.

Notification Hubs limits

TIER	FREE	BASIC	STANDARD
Included pushes	1 million	10 million	10 million
Active devices	500	200,000	10 million
Tag quota per installation or registration	60	60	60

For more information on limits and pricing, see [Notification Hubs pricing](#).

Service Bus limits

The following table lists quota information specific to Azure Service Bus messaging. For information about pricing and other quotas for Service Bus, see [Service Bus pricing](#).

QUOTA NAME	SCOPE	VALUE	NOTES
Maximum number of namespaces per Azure subscription	Namespace	1000 (default and maximum)	Subsequent requests for additional namespaces are rejected.
Queue or topic size	Entity	1, 2, 3, 4 GB or 5 GB. In the Premium SKU, and the Standard SKU with partitioning enabled, the maximum queue or topic size is 80 GB.	Defined upon creation/updation of the queue or topic. Subsequent incoming messages are rejected, and an exception is received by the calling code.

QUOTA NAME	SCOPE	VALUE	NOTES
Number of concurrent connections on a namespace	Namespace	Net Messaging: 1,000. AMQP: 5,000.	Subsequent requests for additional connections are rejected, and an exception is received by the calling code. REST operations don't count toward concurrent TCP connections.
Number of concurrent receive requests on a queue, topic, or subscription entity	Entity	5,000	Subsequent receive requests are rejected, and an exception is received by the calling code. This quota applies to the combined number of concurrent receive operations across all subscriptions on a topic.
Number of topics or queues per namespace	Namespace	10,000 for the Basic or Standard tier. The total number of topics and queues in a namespace must be less than or equal to 10,000. For the Premium tier, 1,000 per messaging unit (MU).	Subsequent requests for creation of a new topic or queue on the namespace are rejected. As a result, if configured through the Azure portal , an error message is generated. If called from the management API, an exception is received by the calling code.
Number of partitioned topics or queues per namespace	Namespace	Basic and Standard tiers: 100. Partitioned entities aren't supported in the Premium tier. Each partitioned queue or topic counts toward the quota of 1,000 entities per namespace.	Subsequent requests for creation of a new partitioned topic or queue on the namespace are rejected. As a result, if configured through the Azure portal , an error message is generated. If called from the management API, the exception QuotaExceededException is received by the calling code. If you want to have more partitioned entities in a basic or a standard tier namespace, create additional namespaces.
Maximum size of any messaging entity path: queue or topic	Entity	-	260 characters.
Maximum size of any messaging entity name: namespace, subscription, or subscription rule	Entity	-	50 characters.

QUOTA NAME	SCOPE	VALUE	NOTES
Maximum size of a message ID	Entity	-	128
Maximum size of a message session ID	Entity	-	128
Message size for a queue, topic, or subscription entity	Entity	Incoming messages that exceed these quotas are rejected, and an exception is received by the calling code.	<p>Maximum message size: 256 KB for Standard tier, 1 MB for Premium tier.</p> <p>Due to system overhead, this limit is less than these values.</p> <p>Maximum header size: 64 KB.</p> <p>Maximum number of header properties in property bag: byte/int.MaxValue.</p> <p>Maximum size of property in property bag: Both the property name and value are limited at 32KB.</p>
Message property size for a queue, topic, or subscription entity	Entity	The exception <code>SerializationException</code> is generated.	<p>Maximum message property size for each property is 32 KB. Cumulative size of all properties can't exceed 64 KB. This limit applies to the entire header of the brokered message, which has both user properties and system properties, such as sequence number, label, and message ID.</p>
Number of subscriptions per topic	Entity	Subsequent requests for creating additional subscriptions for the topic are rejected. As a result, if configured through the portal, an error message is shown. If called from the management API, an exception is received by the calling code.	2,000 per-topic for the Standard tier and Premium tier.
Number of SQL filters per topic	Entity	Subsequent requests for creation of additional filters on the topic are rejected, and an exception is received by the calling code.	2,000

QUOTA NAME	SCOPE	VALUE	NOTES
Number of correlation filters per topic	Entity	Subsequent requests for creation of additional filters on the topic are rejected, and an exception is received by the calling code.	100,000
Size of SQL filters or actions	Namespace	Subsequent requests for creation of additional filters are rejected, and an exception is received by the calling code.	Maximum length of filter condition string: 1,024 (1 K). Maximum length of rule action string: 1,024 (1 K). Maximum number of expressions per rule action: 32.
Number of shared access authorization rules per namespace, queue, or topic	Entity, namespace	Subsequent requests for creation of additional rules are rejected, and an exception is received by the calling code.	Maximum number of rules per entity type: 12. Rules that are configured on a Service Bus namespace apply to all types: queues, topics.
Number of messages per transaction	Transaction	Additional incoming messages are rejected, and an exception stating "Cannot send more than 100 messages in a single transaction" is received by the calling code.	100 For both Send() and SendAsync() operations.
Number of virtual network and IP filter rules	Namespace		128

Site Recovery limits

The following limits apply to Azure Site Recovery.

LIMIT IDENTIFIER	LIMIT
Number of vaults per subscription	500
Number of servers per Recovery Services vault	250
Number of protection groups per Recovery Services vault	No limit
Number of recovery plans per Recovery Services vault	No limit
Number of servers per protection group	No limit
Number of servers per recovery plan	100

SQL Database limits

For SQL Database limits, see [SQL Database resource limits for single databases](#), [SQL Database resource limits for elastic pools and pooled databases](#), and [SQL Database resource limits for SQL Managed Instance](#).

The maximum number of private endpoints per Azure SQL Database logical server is 250.

Azure Synapse Analytics limits

Azure Synapse Analytics has the following default limits to ensure customer's subscriptions are protected from each other's workloads. To raise the limits to the maximum for your subscription, contact support.

Synapse Workspace Limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Synapse workspaces in an Azure subscription	20	20

Synapse Pipeline Limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Synapse pipelines in a Synapse workspace	800	800
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a workspace	5,000	Contact support.
Total CPU cores for Azure-SSIS Integration Runtimes under one workspace	256	Contact support.
Concurrent pipeline runs per workspace that's shared among all pipelines in the workspace	10,000	10,000
Concurrent External activity runs per workspace per Azure Integration Runtime region External activities are managed on integration runtime but execute on linked services, including Databricks, stored procedure, HDInsight, Web, and others. This limit does not apply to Self-hosted IR.	3,000	3,000
Concurrent Pipeline activity runs per workspace per Azure Integration Runtime region Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete. This limit does not apply to Self-hosted IR.	1,000	1,000

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
<p>Concurrent authoring operations per workspace per Azure Integration Runtime region</p> <p>Including test connection, browse folder list and table list, preview data. This limit does not apply to Self-hosted IR.</p>	200	200
<p>Concurrent Data Integration Units¹ consumption per workspace per Azure Integration Runtime region</p>	Region group 1 ² : 6,000 Region group 2 ² : 3,000 Region group 3 ² : 1,500	Region group 1 ² : 6,000 Region group 2 ² : 3,000 Region group 3 ² : 1,500
<p>Maximum activities per pipeline, which includes inner activities for containers</p>	40	40
<p>Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime</p>	100	Contact support.
<p>Maximum parameters per pipeline</p>	50	50
<p>ForEach items</p>	100,000	100,000
<p>ForEach parallelism</p>	20	50
<p>Maximum queued runs per pipeline</p>	100	100
<p>Characters per expression</p>	8,192	8,192
<p>Minimum tumbling window trigger interval</p>	15 min	15 min
<p>Maximum timeout for pipeline activity runs</p>	7 days	7 days
<p>Bytes per object for pipeline objects³</p>	200 KB	200 KB
<p>Bytes per object for dataset and linked service objects³</p>	100 KB	2,000 KB
<p>Bytes per payload for each activity run⁴</p>	896 KB	896 KB
<p>Data Integration Units¹ per copy activity run</p>	256	256
<p>Write API calls</p>	1,200/h	1,200/h This limit is imposed by Azure Resource Manager, not Azure Synapse Analytics.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Read API calls	12,500/h	12,500/h This limit is imposed by Azure Resource Manager, not Azure Synapse Analytics.
Monitoring queries per minute	1,000	1,000
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	Contact support.
Concurrent number of data flow debug sessions per user per workspace	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a workspace	2 GB	Contact support.

¹ The data integration unit (DIU) is used in a cloud-to-cloud copy operation, learn more from [Data integration units \(version 2\)](#). For information on billing, see [Azure Synapse Analytics Pricing](#).

² [Azure Integration Runtime](#) is [globally available](#) to ensure data compliance, efficiency, and reduced network egress costs.

REGION GROUP	REGIONS
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

³ Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Synapse Analytics. Synapse Analytics is designed to scale to handle petabytes of data.

⁴ The payload for each activity run includes the activity configuration, the associated dataset(s) and linked service(s) configurations if any, and a small portion of system properties generated per activity type. Limit for this payload size doesn't relate to the amount of data you can move and process with Azure Synapse Analytics. Learn about the [symptoms and recommendation](#) if you hit this limit.

Dedicated SQL pool limits

For details of capacity limits for dedicated SQL pools in Azure Synapse Analytics, see [dedicated SQL pool resource limits](#).

Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource](#)

Azure Files and Azure File Sync

To learn more about the limits for Azure Files and File Sync, see [Azure Files scalability and performance targets](#).

Storage limits

The following table describes default limits for Azure general-purpose v1, v2, Blob storage, and block blob storage accounts. The *ingress* limit refers to all data that is sent to a storage account. The *egress* limit refers to all data that is received from a storage account.

NOTE

You can request higher capacity and ingress limits. To request an increase, contact [Azure Support](#).

RESOURCE	LIMIT
Number of storage accounts per region per subscription, including standard, and premium storage accounts.	250
Maximum storage account capacity	5 PiB ¹
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account	No limit
Maximum request rate ¹ per storage account	20,000 requests per second
Maximum ingress ¹ per storage account (US, Europe regions)	10 Gbps
Maximum ingress ¹ per storage account (regions other than US and Europe)	5 Gbps if RA-GRS/GRS is enabled, 10 Gbps for LRS/ZRS ²
Maximum egress for general-purpose v2 and Blob storage accounts (all regions)	50 Gbps
Maximum egress for general-purpose v1 storage accounts (US regions)	20 Gbps if RA-GRS/GRS is enabled, 30 Gbps for LRS/ZRS ²
Maximum egress for general-purpose v1 storage accounts (non-US regions)	10 Gbps if RA-GRS/GRS is enabled, 15 Gbps for LRS/ZRS ²
Maximum number of IP address rules per storage account	200
Maximum number of virtual network rules per storage account	200
Maximum number of resource instance rules per storage account	200
Maximum number of private endpoints per storage account	200

¹ Azure Storage standard accounts support higher capacity limits and higher limits for ingress by request. To request an increase in account limits, contact [Azure Support](#).

² If your storage account has read-access enabled with geo-redundant storage (RA-GRS) or geo-zone-redundant storage (RA-GZRS), then the egress targets for the secondary location are identical to those of the primary location. For more information, see [Azure Storage replication](#).

NOTE

Microsoft recommends that you use a general-purpose v2 storage account for most scenarios. You can easily upgrade a general-purpose v1 or an Azure Blob storage account to a general-purpose v2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a general-purpose v2 storage account](#).

All storage accounts run on a flat network topology regardless of when they were created. For more information on the Azure Storage flat network architecture and on scalability, see [Microsoft Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](#).

For more information on limits for standard storage accounts, see [Scalability targets for standard storage accounts](#).

Storage resource provider limits

The following limits apply only when you perform management operations by using Azure Resource Manager with Azure Storage.

RESOURCE	LIMIT
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	10 per second / 1200 per hour
Storage account management operations (list)	100 per 5 minutes

Azure Blob storage limits

RESOURCE	TARGET
Maximum size of single blob container	Same as maximum storage account capacity
Maximum number of blocks in a block blob or append blob	50,000 blocks
Maximum size of a block in a block blob	4000 MiB
Maximum size of a block blob	50,000 X 4000 MiB (approximately 190.7 TiB)
Maximum size of a block in an append blob	4 MiB
Maximum size of an append blob	50,000 x 4 MiB (approximately 195 GiB)
Maximum size of a page blob	8 TiB ²
Maximum number of stored access policies per blob container	5
Target request rate for a single blob	Up to 500 requests per second
Target throughput for a single page blob	Up to 60 MiB per second ²

RESOURCE	TARGET
Target throughput for a single block blob	Up to storage account ingress/egress limits ¹

¹ Throughput for a single blob depends on several factors, including, but not limited to: concurrency, request size, performance tier, speed of source for uploads, and destination for downloads. To take advantage of the performance enhancements of [high-throughput block blobs](#), upload larger blobs or blocks. Specifically, call the [Put Blob](#) or [Put Block](#) operation with a blob or block size that is greater than 4 MiB for standard storage accounts. For premium block blob or for Data Lake Storage Gen2 storage accounts, use a block or blob size that is greater than 256 KiB.

² Page blobs are not yet supported in accounts that have the **Hierarchical namespace** setting on them.

The following table describes the maximum block and blob sizes permitted by service version.

SERVICE VERSION	MAXIMUM BLOCK SIZE (VIA PUT BLOCK)	MAXIMUM BLOB SIZE (VIA PUT BLOCK LIST)	MAXIMUM BLOB SIZE VIA SINGLE WRITE OPERATION (VIA PUT BLOB)
Version 2019-12-12 and later	4000 MiB	Approximately 190.7 TiB (4000 MiB X 50,000 blocks)	5000 MiB (preview)
Version 2016-05-31 through version 2019-07-07	100 MiB	Approximately 4.75 TiB (100 MiB X 50,000 blocks)	256 MiB
Versions prior to 2016-05-31	4 MiB	Approximately 195 GiB (4 MiB X 50,000 blocks)	64 MiB

Azure Queue storage limits

RESOURCE	TARGET
Maximum size of a single queue	500 TiB
Maximum size of a message in a queue	64 KiB
Maximum number of stored access policies per queue	5
Maximum request rate per storage account	20,000 messages per second, which assumes a 1-KiB message size
Target throughput for a single queue (1-KiB messages)	Up to 2,000 messages per second

Azure Table storage limits

The following table describes capacity, scalability, and performance targets for Table storage.

RESOURCE	TARGET
Number of tables in an Azure storage account	Limited only by the capacity of the storage account
Number of partitions in a table	Limited only by the capacity of the storage account
Number of entities in a partition	Limited only by the capacity of the storage account

RESOURCE	TARGET
Maximum size of a single table	500 TiB
Maximum size of a single entity, including all property values	1 MiB
Maximum number of properties in a table entity	255 (including the three system properties, PartitionKey , RowKey , and Timestamp)
Maximum total size of an individual property in an entity	Varies by property type. For more information, see Property Types in Understanding the Table Service Data Model .
Size of the PartitionKey	A string up to 1 KiB in size
Size of the RowKey	A string up to 1 KiB in size
Size of an entity group transaction	A transaction can include at most 100 entities and the payload must be less than 4 MiB in size. An entity group transaction can include an update to an entity only once.
Maximum number of stored access policies per table	5
Maximum request rate per storage account	20,000 transactions per second, which assumes a 1-KiB entity size
Target throughput for a single table partition (1 KiB-entities)	Up to 2,000 entities per second

Virtual machine disk limits

You can attach a number of data disks to an Azure virtual machine. Based on the scalability and performance targets for a VM's data disks, you can determine the number and type of disk that you need to meet your performance and capacity requirements.

IMPORTANT

For optimal performance, limit the number of highly utilized disks attached to the virtual machine to avoid possible throttling. If all attached disks aren't highly utilized at the same time, the virtual machine can support a larger number of disks.

For Azure managed disks:

The following table illustrates the default and maximum limits of the number of resources per region per subscription. The limits remain the same irrespective of disks encrypted with either platform-managed keys or customer-managed keys. There is no limit for the number of Managed Disks, snapshots and images per resource group.

RESOURCE	LIMIT
Standard managed disks	50,000
Standard SSD managed disks	50,000
Premium managed disks	50,000

RESOURCE	LIMIT
Standard_LRS snapshots	50,000
Standard_ZRS snapshots	50,000
Managed image	50,000

For Standard storage accounts: A Standard storage account has a maximum total request rate of 20,000 IOPS. The total IOPS across all of your virtual machine disks in a Standard storage account should not exceed this limit.

You can roughly calculate the number of highly utilized disks supported by a single Standard storage account based on the request rate limit. For example, for a Basic tier VM, the maximum number of highly utilized disks is about 66, which is 20,000/300 IOPS per disk. The maximum number of highly utilized disks for a Standard tier VM is about 40, which is 20,000/500 IOPS per disk.

For Premium storage accounts: A Premium storage account has a maximum total throughput rate of 50 Gbps. The total throughput across all of your VM disks should not exceed this limit.

For more information, see [Virtual machine sizes](#).

Disk encryption sets

There's a limitation of 1000 disk encryption sets per region, per subscription. For more information, see the encryption documentation for [Linux](#) or [Windows](#) virtual machines. If you need to increase the quota, contact Azure support.

Managed virtual machine disks

Standard HDD managed disks

STANDARD DISK TYPE	S4	S6	S10	S15	S20	S30	S40	S50	S60	S70	S80
Disk size in GiB	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
IOPS per disk	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 1,300	Up to 2,000	Up to 2,000
Throughput per disk	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 300 MB/sec	Up to 500 MB/sec	Up to 500 MB/sec

Standard SSD managed disks

STANDARD SSD SIZES	E1	E2	E3	E4	E6	E10	E15	E20	E30	E40	E50	E60	E70	E80
Disk size in GiB	4	8	16	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
IOPS per disk	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 500	Up to 2,000	Up to 4,000	Up to 6,000
Throughput per disk	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 60 MB/sec	Up to 400 MB/sec	Up to 600 MB/sec	Up to 750 MB/sec
Max burst IOPS per disk	600	600	600	600	600	600	600	600	1000					
Max burst throughput per disk	150 MB/sec	150 MB/sec	150 MB/sec	150 MB/sec	150 MB/sec	150 MB/sec	150 MB/sec	150 MB/sec	250 MB/sec					
Max burst duration	30 min	30 min	30 min	30 min	30 min	30 min	30 min	30 min	30 min					

Premium SSD managed disks: Per-disk limits

PREMIUM SSD SIZES	P1	P2	P3	P4	P6	P10	P15	P20	P30	P40	P50	P60	P70	P80
Eligible for reservation	No	No	No	No	No	No	No	No	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year

*Applies only to disks with on-demand bursting enabled.

Premium SSD managed disks: Per-VM limits

RESOURCE	LIMIT
Maximum IOPS Per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/s with GS5 VM

Unmanaged virtual machine disks

Standard unmanaged virtual machine disks: Per-disk limits

VM TIER	BASIC TIER VM	STANDARD TIER VM
Disk size	4,095 GB	4,095 GB
Maximum 8-KB IOPS per persistent disk	300	500
Maximum number of disks that perform the maximum IOPS	66	40

Premium unmanaged virtual machine disks: Per-account limits

RESOURCE	LIMIT
Total disk capacity per account	35 TB
Total snapshot capacity per account	10 TB
Maximum bandwidth per account (ingress + egress) ¹	<=50 Gbps

¹*Ingress* refers to all data from requests that are sent to a storage account. *Egress* refers to all data from responses that are received from a storage account.

Premium unmanaged virtual machine disks: Per-disk limits

PREMIUM STORAGE DISK TYPE	P10	P20	P30	P40	P50
Disk size	128 GiB	512 GiB	1,024 GiB (1 TB)	2,048 GiB (2 TB)	4,095 GiB (4 TB)

PREMIUM STORAGE DISK TYPE	P10	P20	P30	P40	P50
Maximum IOPS per disk	500	2,300	5,000	7,500	7,500
Maximum throughput per disk	100 MB/sec	150 MB/sec	200 MB/sec	250 MB/sec	250 MB/sec
Maximum number of disks per storage account	280	70	35	17	8

Premium unmanaged virtual machine disks: Per-VM limits

RESOURCE	LIMIT
Maximum IOPS per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/sec with GS5 VM

StorSimple System limits

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of storage account credentials	64	
Maximum number of volume containers	64	
Maximum number of volumes	255	
Maximum number of schedules per bandwidth template	168	A schedule for every hour, every day of the week.
Maximum size of a tiered volume on physical devices	64 TB for StorSimple 8100 and StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum size of a tiered volume on virtual devices in Azure	30 TB for StorSimple 8010 64 TB for StorSimple 8020	StorSimple 8010 and StorSimple 8020 are virtual devices in Azure that use Standard storage and Premium storage, respectively.
Maximum size of a locally pinned volume on physical devices	9 TB for StorSimple 8100 24 TB for StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum number of iSCSI connections	512	
Maximum number of iSCSI connections from initiators	512	

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of access control records per device	64	
Maximum number of volumes per backup policy	24	
Maximum number of backups retained per backup policy	64	
Maximum number of schedules per backup policy	10	
Maximum number of snapshots of any type that can be retained per volume	256	This amount includes local snapshots and cloud snapshots.
Maximum number of snapshots that can be present in any device	10,000	
Maximum number of volumes that can be processed in parallel for backup, restore, or clone	16	<ul style="list-style-type: none"> • If there are more than 16 volumes, they're processed sequentially as processing slots become available. • New backups of a cloned or a restored tiered volume can't occur until the operation is finished. For a local volume, backups are allowed after the volume is online.

LIMIT IDENTIFIER	LIMIT	COMMENTS
Restore and clone recover time for tiered volumes	<2 minutes	<ul style="list-style-type: none"> • The volume is made available within 2 minutes of a restore or clone operation, regardless of the volume size. • The volume performance might initially be slower than normal as most of the data and metadata still resides in the cloud. Performance might increase as data flows from the cloud to the StorSimple device. • The total time to download metadata depends on the allocated volume size. Metadata is automatically brought into the device in the background at the rate of 5 minutes per TB of allocated volume data. This rate might be affected by Internet bandwidth to the cloud. • The restore or clone operation is complete when all the metadata is on the device. • Backup operations can't be performed until the restore or clone operation is fully complete.

LIMIT IDENTIFIER	LIMIT	COMMENTS
Restore recover time for locally pinned volumes	<2 minutes	<ul style="list-style-type: none"> The volume is made available within 2 minutes of the restore operation, regardless of the volume size. The volume performance might initially be slower than normal as most of the data and metadata still resides in the cloud. Performance might increase as data flows from the cloud to the StorSimple device. The total time to download metadata depends on the allocated volume size. Metadata is automatically brought into the device in the background at the rate of 5 minutes per TB of allocated volume data. This rate might be affected by Internet bandwidth to the cloud. Unlike tiered volumes, if there are locally pinned volumes, the volume data is also downloaded locally on the device. The restore operation is complete when all the volume data has been brought to the device. The restore operations might be long and the total time to complete the restore will depend on the size of the provisioned local volume, your Internet bandwidth, and the existing data on the device. Backup operations on the locally pinned volume are allowed while the restore operation is in progress.
Thin-restore availability	Last failover	
Maximum client read/write throughput, when served from the SSD tier*	920/720 MB/sec with a single 10-gigabit Ethernet network interface	Up to two times with MPIO and two network interfaces.
Maximum client read/write throughput, when served from the HDD tier*	120/250 MB/sec	
Maximum client read/write throughput, when served from the cloud tier*	11/41 MB/sec	Read throughput depends on clients generating and maintaining sufficient I/O queue depth.

*Maximum throughput per I/O type was measured with 100 percent read and 100 percent write scenarios. Actual throughput might be lower and depends on I/O mix and network conditions.

Stream Analytics limits

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of streaming units per subscription per region	500	To request an increase in streaming units for your subscription beyond 500, contact Microsoft Support .
Maximum number of inputs per job	60	There's a hard limit of 60 inputs per Azure Stream Analytics job.
Maximum number of outputs per job	60	There's a hard limit of 60 outputs per Stream Analytics job.
Maximum number of functions per job	60	There's a hard limit of 60 functions per Stream Analytics job.
Maximum number of streaming units per job	192	There's a hard limit of 192 streaming units per Stream Analytics job.
Maximum number of jobs per region	1,500	Each subscription can have up to 1,500 jobs per geographical region.
Reference data blob MB	5 GB	Up to 5 GB when using 6 SUs or more.
Maximum number of characters in a query	512000	There's a hard limit of 512k characters in an Azure Stream Analytics job query.

Virtual Machines limits

Virtual Machines limits

RESOURCE	LIMIT
Virtual machines per cloud service ¹	50
Input endpoints per cloud service ²	150

¹ Virtual machines created by using the classic deployment model instead of Azure Resource Manager are automatically stored in a cloud service. You can add more virtual machines to that cloud service for load balancing and availability.

² Input endpoints allow communications to a virtual machine from outside the virtual machine's cloud service. Virtual machines in the same cloud service or virtual network can automatically communicate with each other.

Virtual Machines limits - Azure Resource Manager

The following limits apply when you use Azure Resource Manager and Azure resource groups.

RESOURCE	LIMIT
VMs per subscription	25,000 ¹ per region.
VM total cores per subscription	20 ¹ per region. Contact support to increase limit.

RESOURCE	LIMIT
Azure Spot VM total cores per subscription	20 ¹ per region. Contact support to increase limit.
VM per series, such as Dv2 and F, cores per subscription	20 ¹ per region. Contact support to increase limit.
Availability sets per subscription	2,500 per region.
Virtual machines per availability set	200
Proximity placement groups per resource group	800
Certificates per availability set	199 ²
Certificates per subscription	Unlimited ³

¹ Default limits vary by offer category type, such as Free Trial and Pay-As-You-Go, and by series, such as Dv2, F, and G. For example, the default for Enterprise Agreement subscriptions is 350. For security, subscriptions default to 20 cores to prevent large core deployments. If you need more cores, submit a support ticket.

² Properties such as SSH public keys are also pushed as certificates and count towards this limit. To bypass this limit, use the [Azure Key Vault extension for Windows](#) or the [Azure Key Vault extension for Linux](#) to install certificates.

³ With Azure Resource Manager, certificates are stored in the Azure Key Vault. The number of certificates is unlimited for a subscription. There's a 1-MB limit of certificates per deployment, which consists of either a single VM or an availability set.

NOTE

Virtual machine cores have a regional total limit. They also have a limit for regional per-size series, such as Dv2 and F. These limits are separately enforced. For example, consider a subscription with a US East total VM core limit of 30, an A series core limit of 30, and a D series core limit of 30. This subscription can deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores. An example of a combination is 10 A1 VMs and 20 D1 VMs.

Shared Image Gallery limits

There are limits, per subscription, for deploying resources using Shared Image Galleries:

- 100 shared image galleries, per subscription, per region
- 1,000 image definitions, per subscription, per region
- 10,000 image versions, per subscription, per region

Virtual machine scale sets limits

RESOURCE	LIMIT
Maximum number of VMs in a scale set	1,000
Maximum number of VMs based on a custom VM image in a scale set	600
Maximum number of scale sets in a region	2,500

See also

- [Understand Azure limits and increases](#)
- [Virtual machine and cloud service sizes for Azure](#)
- [Sizes for Azure Cloud Services](#)
- [Naming rules and restrictions for Azure resources](#)